Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2016

José Francisco Rodrigues Sequeira

Construção e enriquecimento de redes de conceitos a partir de texto livre

Automatic Knowledge Base Construction from unstructured text

Universidade de Aveiro
2016

Departamento de Eletrónica,
Telecomunicações e Informática

José Francisco Rodrigues Sequeira

Construção e enriquecimento de redes de conceitos a partir de texto livre

Automatic Knowledge Base Construction from unstructured text

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Sérgio Guilherme Aleixo de Matos, Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Antes de mais, gostaria de agradecer ao meu orientador, Sérgio Matos, pela sua orientação, tanto nesta tese como nos anos passados como investigador pelo grupo de Bioinformática do IEETA. Sinto que as oportunidades que me foram proporcionadas e os desafios que me lançaram foram fulcrais para o meu crescimento como profissional, na busca dos meus interesses profissionais e na procura de melhorar as minhas qualidades.

Quero agradecer à minha mãe e aos meus irmãos, que me acompanharam/aturaram estes anos todos e que nunca me deixaram duvidar das minhas capacidades. Faço um agradecimento especial à minha mãe, o maior exemplo de dedicação que tive na vida, porque sem a sua educação, amor e apoio que recebi, nada disto seria possível.

Agradeço aos meus colegas e amigos que me acompanharam neste curso, Miguel Vicente, Vasco Santos, Joel Pinheiro, Rui Monteiro, André Jerónimo, José Mendes e João Abel por fazerem com que este percurso fizesse sentido e por me mostrarem o quão fácil é ultrapassar obstáculos quando se está rodeado de pessoas excepcionais. O meu percurso por Aveiro termina mas levo-vos comigo.

Ao meu curso, entidade enorme que é. Obrigado por terem trazido ao de cima o melhor de mim.

Aos meus amigos de infância, César, Roque e Vasco, por terem-me mostrado que 5 anos de distância nada querem dizer. Ao sucesso, meus amigos!

Um agradecimento especial à Beatriz França, que teve de aguentar a minha pessoa num dos anos mais stressantes da minha vida. Se não estivesses lá para me dar força todos os dias, sinto que este documento não seria uma realidade.

Obrigado Aveiro, sem dúvida tens mais encanto na hora da despedida.

o júri / the jury

presidente / president

Prof. Dr. José Luis Oliveira

Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Dr. Sérgio Guilherme Aleixo de Matos

Investigador Auxiliar do Departamento Engenharia e Telecomunicações da Universidade de Aveiro

Prof. Dr. Carla Teixeira Lopes

Professor auxiliar da Faculdade de Engenharia da Universidade do Porto

Palavras Chave

Bioinformática, Mineração de Texto, Associação de Palavras, Visualização de Dados, Recuperação de Informação, Armazenamento de Dados, Reconhecimento de Conceitos.

Resumo

Tendo em conta o crescimento do número de publicações biomédicas a serem produzidas todos os anos, o esforço exigido para que um utilizador consiga, de uma forma eficiente, explorar estas publicações para conseguir estabelecer associações entre um conjunto alargado de conceitos torna esta tarefa exaustiva.

Nesta disertação apresentamos uma plataforma web chamada GRACE, que providencia uma interface gráfica de exploração que permite aos utilizadores navegar pelo domínio biomédico em busca de associações explícitas ou latentes entre conceitos biomédicos pertencentes a uma variedade de domínios semânticos (i.e., Genes, Proteínas, Doenças, Procedimentos e Anatomia). A base de conhecimento usada é uma coleção de artigos MEDLINE com resumos escritos na língua inglesa. Estas anotações são armazenadas numa base de dados que permite pesquisas complexas e obtenção de dados com alta performance. As relações entre conceitos são inferidas a partir de análise estatística, aplicando medidas de associações entre os conceitos anotados. Estes processos permitem à interface gráfica criar, em tempo real, uma visualização de dados, na forma de um grafo, para a exploração destas relações entre conceitos do domínio biomédico.

Keywords Bioinformatics, Text Mining, Word Assocation, Data Visualization, Information Retrieval, Data Storage, Concept Recognition.

Abstract

Taking into account the overwhelming number of biomedical publications being produced, the effort required for a user to efficiently explore those publications in order to establish relationships between a wide range of concepts is staggering.

This dissertation presents GRACE, a web-based platform that provides an advanced graphical exploration interface that allows users to traverse the biomedical domain in order to find explicit and latent associations between annotated biomedical concepts belonging to a variety of semantic types (e.g., Genes, Proteins, Disorders, Procedures and Anatomy). The knowledge base utilized is a collection of MEDLINE articles with English abstracts. These annotations are then stored in an efficient data storage that allows for complex queries and high-performance data delivery. Concept relationship are inferred through statistical analysis, applying association measures to annotated terms. These processes grant the graphical interface the ability to create, in real-time, a data visualization in the form of a graph for the exploration of these biomedical concept relationships.

Contents

List of Figures

List of Tables

Listings

Acronyms

DM	Data Mining
TM	Text Mining
IR	Information Retrieval
IE	Information Extraction
MUC	Message Understanding Conferences
NER	Named Entity Recognition
NLP	Natural Language Processing
ML	Machine-Learning
ROI	regions of interest
POS	part-of-speech
TSV	tab-separated values
WSD	Word Sense Disambiguation
tf-idf	term frequency–inverse document frequency
PDF	Portable Document Format
DSL	Domain Specific Language
NRT	Near Real Time
JSON	JavaScript Object Notation
CRF	Conditional Random Fields
NCBI	National Center for Biotechnology Information
MESH	Medical Subject Headings
NLM	National Library of Medicine
PII	Publisher Item Identifier
DOI	Digital Object Identifier
CUI	Concept Unique Identifier
ACID	Atomicity, Consistency, Isolation, Durability
DBMS	Database Management System
JVM	Java Virtual Machine
UIMA	Unstructured Information Management Architecture
SemMedBD	Semantic Medline Database
PMI	Pointwise Mutual Information
UMLS	Unified Medical Language System
NLM	National Library of Medicine

CHAPTER 1

Introduction

The desire to store and share information has been a part of human nature, at least for the majority of the civilizations in the planet, for thousands of years [1]. The ability to preserve information and being able to retrieve it has been used as a source of knowledge since ancient times, allowing us to share valuable data. We resorted to physical objects to store that information, however we often lacked the ability to efficiently share those objects with each other. Gutenberg's work on the printing press overcame that issue, increasing the amount of books available and making them affordable by a bigger portion of the population. As books became a standard for the representation of knowledge, we developed the need to categorize, grouping them by author, subject, year, genre.

With the evolution of computer science, the sheer amount of available data has increased in a way that seemed unthinkable. The Internet truly revolutionized the way our society handles data, since it has become broadly available in various formats (text, videos, sound and images). Access to these files has been made trivial by the evolution of the technological devices we use to consume them. Nowadays every event, article or social media feed is stored for analysis, reference or just for the sake of keeping said information.

These new sources of information raised a new problem for data management since unlike previously stored data, which had a high degree of organization and was stored in relational databases, there was no common pattern that would fit resources originated from so many distinct sources. This type of data, that has no pre-defined model or pattern is called `unstructured data` [2].

This increase in available data has encouraged scientific research to develop tools that are able to obtain, access and filter this information effectively. However, the lack of structure inherent to the storage of these files makes the procedure of processing, filtering and interpretation of this data difficult. There has been a high interest in the development of techniques that can identify, extract, manage, integrate and exploit the information present in these files. This dissertation focuses on data presented in the form of text, which composes a large amount of the available data.

1.1 THE INFORMATION RETRIEVAL CHALLENGE

Text Mining (TM) is the area of computer science, under the field of Data Mining (DM), that addresses the challenges of searching and identifying implicit patterns and trends in documents [3], extracting information that is relevant and understanding the meaning of the information retrieved. The development of efficient techniques that solve these problems allows the retrieval of high quality information from free-text, represent it in a structured form and inducing knowledge from the extracted unstructured data.

The main objectives of text mining are divided into Information Retrieval (IR), which aims to identify and obtain data sources that match a user's information needs [4] and Information Extraction (IE) with the goal of populating a structured data storage from an unstructured data source [5].

The Message Understanding Conferences (MUC) [6] defined several tasks that must be accomplished in order to complete the IE goals:

Named Entity Recognition (NER)
The ability to identify parts of the text as specific entity names, ranging from concepts, people, places and organizations. This allows for portions of the text to be identified as concepts so they can be normalized and mapped to a real concept.

Normalization and Disambiguation
The association of a concept to a unique meaning. This task is straightforward for concepts which have only one meaning (e.g., a unique date such as "June 6, 1944"), however in some cases there is the need for disambiguation (e.g., "Titus Andronicus" may refer to a Shakespeare play or to a band name).

Coreference
Being able to associate two different associations to the same concept (e.g the association of "it" to a concept present, normally in the same sentence).

Relation Mining
The possibility of extracting a relationship between two different concepts (e.g., in the sentence "Smoking increases the probability of developing lung cancer" there is an explicit relationship smoking $\rightarrow$ lung cancer).

Summarization
Being able to extract the relevant information from a text or a collection of texts and presenting it in a text with shorter length.

Classification
Associating a text (or portion) to a specific theme (e.g., music, sports, medicine).

In the context of this dissertation, we will mainly focus on the task of NER and normalization, using the results obtained to create a rich, concept-driven, queryable infrastructure.

1.2 OBJECTIVE

Nowadays the amount of available data on the biomedical field has become very large, Medline [7], the National Library of Medicine journal citation database, hosts over 22 million scientific publications in this area. The effort required by a human being to scavenge through this information to fulfill his information needs may be overwhelming. Keeping this premise in mind, the work described in this dissertation aims to overcome the challenge of reliably extracting and representing, in an explicit and user-friendly application, relationships between biomedical concepts. As a result, a web application was developed so that users are able to visualize concept relationship graphs, based on annotated biomedical concepts found in the MedLine *corpus*, by executing queries on the client app. The data was annotated by a Named Entity Recognition framework and stored using an advanced indexing platform. Relationships between concepts were established based on the co-occurrence of distinct concepts.

CHAPTER 2

Background

This section focuses on the study of concepts and tools relevant to the objective of this dissertation, it provides a detailed overview of several areas that are intrinsic to the final product's applicability.

The knowledge source for this work will be the biomedical literature database *Medline*, which is composed by over 22 million citations and around 14 million abstracts of biomedical scientific journal articles. This work has the purpose of creating a structure that will have a semantic indexing of the data available on the database, establishing associations between concepts, based on their co-occurrences in the collection. This, however, poses some obstacles.

First we must perform the task of NER on the data set with the objective of identifying important biomedical concepts, since in this work the basic unit of information will be the concepts identified by the IE tool used.

The task of disambiguation of medical concepts poses a challenge in the task of IE, since it is frequent for different concepts to have equal names. For instance, it is common for a gene, a protein encoded by the gene and a disease associated with the protein to have the same name.

Establishing relationships between concepts in medical data is a task with many layers of complexity and can be approached in various ways. It may be done through binary relations (e.g., protein-protein interaction) or utilizing Natural Language Processing (NLP) to create cause-effect semantic associations between concepts.

2.1 Information extraction in biomedical text

2.1.1 Named Entity Recognition in biomedical data

The objective of NER is to establish a link between a chunk of text and a specific concept of interest. The most common approaches for this type of recognition are rule-based, dictionary-based

and using Machine-Learning (ML) techniques. The generic pipeline of such a framework [8] is described in figure 2.1, and is composed of the following processes:

Reader Reader and parser for document collections in the form of raw text.

Pre-Processing A set of NLP actions that allow tokenization and enable the recognition process.

NER Automatically associate a token or a sequence of tokens to a specific concept.

Post-Processing Refinement of the concepts identified.

Writer Structured representation of concepts identified in input Corpus.

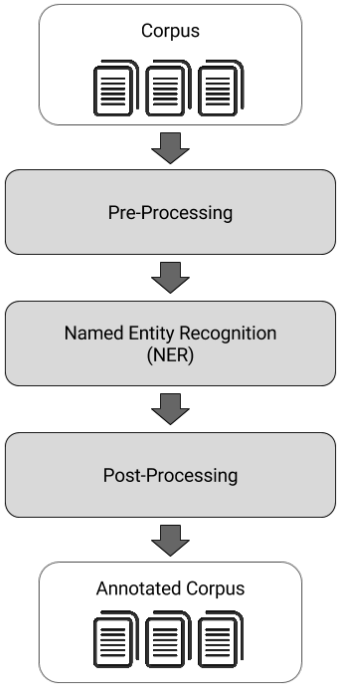

Figure 2.1: Generic NER framework pipeline

Although all the approaches enumerated above use a similar pipeline, the *corpus* can have many different characteristics and so the approach chosen for the task of NER should be decided taking into account the specifications of the original corpus and the type of concepts that need to be identified. In biomedical text mining it is necessary to choose an approach that better fits the type of concept to be identified:

Rule-Based Should be used to find names that follow a strongly defined morphological structure.

Dictionary-Based Should be used in cases where the named entities belong to a well defined set of names that can be inserted in a collection (e.g., animal species, Cities in Europe).

Machine-Learning Based Should be used on the identification of concepts that appear in different lexical variations and/or that have a dynamic set of names (e.g., genes, proteins and chemicals).

Each one of these approaches has different technical specifications and those should also be taken into account:

Rule-Based

Rule-based approaches combine orthographic patterns with lexical analysis and word syntax, based on criteria specified by experts on the area. This type of systems works very well when they are developed for a specific corpus in order to identify concepts that follow the defined rules. This makes rules very specific to the data set they were created for. Therefore, when applied in a different context, there is generally a noticeable performance drop, since the strategy is not very versatile. It is an approach recommended for the recognition of a strictly defined set of concepts that follow a certain pattern.

Dictionary-Based

Dictionaries contain a set of names that reliably represent a certain domain. This strategy tries to establish a match between a dictionary entry and chunks of the unstructured text. Entries found by dictionary matching are associated with a unique identifier from a curated knowledge base, these will make the correspondence between a chunk of text and a specific concept. Nonetheless, this approach presents certain problems:

- There is usually a large number of false positives, especially caused by shorter concept names.
- There is the possibility of a concept having spelling variations. If exact matching is used all matches must be identical, therefore all occurrences that have a spelling that was not taken into account by the curator of the dictionary, will be disregarded.
- When using a large number of dictionaries, there is the possibility of the same word being represented in more than one of them. This creates the need for disambiguation of these concepts.

ML-Based

A ML approach consists of training a computational model to induce the characteristics of specific entity names and is composed of two distinct phases: training and annotating. For the training step, the model is trained to recognize entity names by analyzing text that has been previously annotated by curators. After the training, the model is ready to be used on non-annotated text, attempting to predict chunks of text that are likely, according to a probability score, to be entity names. This approach is broader than a dictionary-based one since it is able to recognize spelling variations. However, it does not provide an identifier for the identified concepts. Usually, there is a post-processing step, using a strategy that associates chunks of identified entities with dictionary entries, trough exact or approximate matching, in order to obtain the corresponding identifier, concluding the normalization process.

2.1.2 BIOMEDICAL CONCEPT ANNOTATORS

There are several systems available for the task of annotations of biomedical concepts such as: MetaMap [9], NCBO Annotator [10], ConceptMapper [11] and Neji [12].

METAMAP

MetaMap [9] is biomedical text annotation tool that uses the Unified Medical Language System (UMLS) [13] Methathesaurus and rule-based annotations in order to retrieve chunks of text. These chunks are scored based on their probability of being a candidate for an entity name.

This type of approach however, has been proven not as efficient as dictionary matching or machine learning solutions since both of these solutions provide better results.

Furthermore, the fact that MetaMap presents itself as an end-user application limits its configurability and adaptability.

NCBO ANNOTATOR

The NCBO Annotator [10] tool presents itself as a web service, providing a platform that annotates text based on the matching of ontology named entities gathered from UMLS and BioPortal [14].

NCBO utilizes dictionary matching to execute the task of NER. Afterwards, a Semantic Expansion is applied to each identified concept based on ontology mappings, relations defined in UMLS and semantic similarity algorithms.

Although NCBO provides a trustworthy and swift solution for concept annotation it presents two limitations: **a)** it does not provide a solution for concept disambiguation causing it to present all identified concepts, including overlapping annotations; **b)** it does not support Machine Learning solutions, which could improve annotation results.

CONCEPTMAPPER

ConceptMapper [11] is a highly configurable, high performance dictionary lookup tool, implemented as a Unstructured Information Management Architecture (UIMA) component. Using one of several matching algorithms, it maps entries in a dictionary onto input documents, producing UIMA annotations.

Individual dictionary entries can contain multiple terms (tokens), and ConceptMapper can be configured to allow multi-term entries to be matched against non-contiguous text. It is also designed to perform fast and has been able to provide real-time results even with multi-million entry dictionaries.

Lookups are token-based and are limited to a specific context, usually a sentence, though this is configurable (e.g., a noun phrase, a paragraph or other NLP-based concept).

2.1.3 NEJI

The tool used for NER will be Neji [12], a modular framework for biomedical concept recognition. Neji has the ability to use a dictionary-based and/or a ML-based solution to solve the task of NER. Our approach will be mainly dictionary-based. Figure 2.2 shows the modular processing pipeline architecture of Neji.

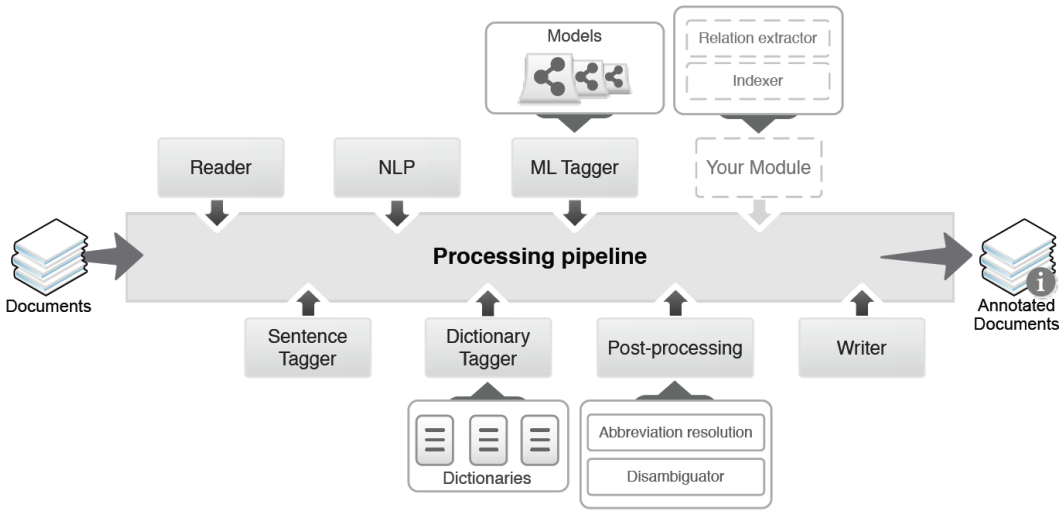

Figure 2.2: Neji's processing pipeline[12]

PIPELINE MODULES

Reader

Used to interpret the input data, filtering out the relevant data and converting it to a format that is compatible with the following modules. The text is divided into regions of interest (ROI).

NLP

This module performs sentence-splitting on the text of each ROI, creating the basic unit of the following modules: the sentence. It performs the NLP task using the GDep [15] tool, a dependency parser for the biomedical domain, which applies tokenization, lemmatization, part-of-speech (POS) tagging, chunking and dependency parsing. When using only dictionary matching just the tokenization resource is utilized.

Dictionary Tagger

One or more dictionaries must be added to the pipeline to execute NER by dictionary matching. Each semantic group should be in a separate file in the tab-separated values (TSV) format. In order to minimize the number of overlapping matches and optimize results the following are applied: *a)* if there is only one dictionary, only the entry with the largest span is considered; *b)* if two dictionary entries with same text exist, then both entries are considered, resulting in the association to two identifiers.

ML Tagger

Neji has the ability to create and train ML models, using the Conditional Random Fields (CRF) [16] implementation from MALLET [17]. There is the possibility to specify the model's configuration and so different models may be trained by Neji. Each model is usually assigned to a specific biomedical concept type.

NER is performed according to a probabilistic model in which sequential tokens with a score above a threshold are identified as an entity mention. However, this does not complete the task of associating a chunk of text to a unique identifier; this normalization step is resolved using dictionary matching to map recognized entity to concept identifiers.

Post-Processing

Neji integrates a post-processing module, designed to complete two different tasks. One of them is finding abbreviated forms of concepts, resorting to pattern-matching rules [18] to extend the annotated full forms.

It also has the task of removing annotations that overlap, according to some rules: *a)* Depth: remove the annotations with a greater depth, *b)* Same group Nested: remove annotations that are contained in larger annotations of the same semantic group, *c)* Priority: Remove annotations that rank lower in a prioritized list.

Writer

The writer is responsible for transforming the annotated set contained in Neji into the desired output format. Neji supports the following formats: A1 [19], CoNLL [20], JSON [21] and allows for the integration of custom writers.

2.1.4 NORMALIZATION AND WORD SENSE DISAMBIGUATION

The process of Named Entity Recognition is followed by the process of normalization and disambiguation. The goal of this procedure is to associate a unique identifier from a curated knowledge source to each chunk of identified text.

Typically a dictionary-based approach is used, mapping dictionary entries to chunks of text previously recognized as entity names. This approach differs slightly from the approach used

in dictionary-based NER since matching can be more flexible, resorting to regular expression or approximate matching approaches.

The association between entity names and dictionary entries can have three different outcomes, as described below:

- If a named entity has only one associated identifier, it is directly assigned to it, ending the normalization process;
- If a named entity has no associated identifier, it is generally discarded;
- If a named association can be associated to a number of different identifiers then the concept is considered ambiguous. Ambiguity is common in the biomedical domain [22].

The existence of ambiguous terms leads to the need of solutions that can correctly disambiguate these terms and associate them to the correct identifier. This process is named Word Sense Disambiguation (WSD) [23] and when successfully performed there is an increase on the number of normalized concepts which usually leads to better concept recognition. Summarizing, the process of WSD aims to reduce the number of ambiguous entity names by attempting to determine its correct meaning in a specific context, resorting to a number of different resources [24].

In the biomedical field, several resources have been made available to facilitate the task of WSD, such as NLM WSD [25], containing over 30 thousand annotated MEDLINE abstracts based on articles of the year 2010 and MedStract [26], which contains over 7000 annotated abstracts.

2.2 INFORMATION RETRIEVAL

Information Retrieval is an ample field in the area of text mining that enables research on technology that can improve the ability to deal with the representation, storage, organization of, and access to information items [27]. Its main purpose is creating a structure that enables users to access information in an efficient manner, with the focus in the information needs of the user. The generic IR process is described in Figure 2.3, which consists of a sequence of processes:

Indexing
Responsible for the correct storage and organization of the data, allowing quick access to the stored information.

Query
Filled out by the user, determines what information will be requested on the index. Results are expected to be related to the inquiry.

Searching
In charge of retrieving information from the index, generally meeting criteria previously defined by a user query.

Ranking
An optional task, however, it is so important that it is implemented by almost every single IR

platform. Allows the results to be sorted, based on a heuristic approach, according to their relevance to the user query.

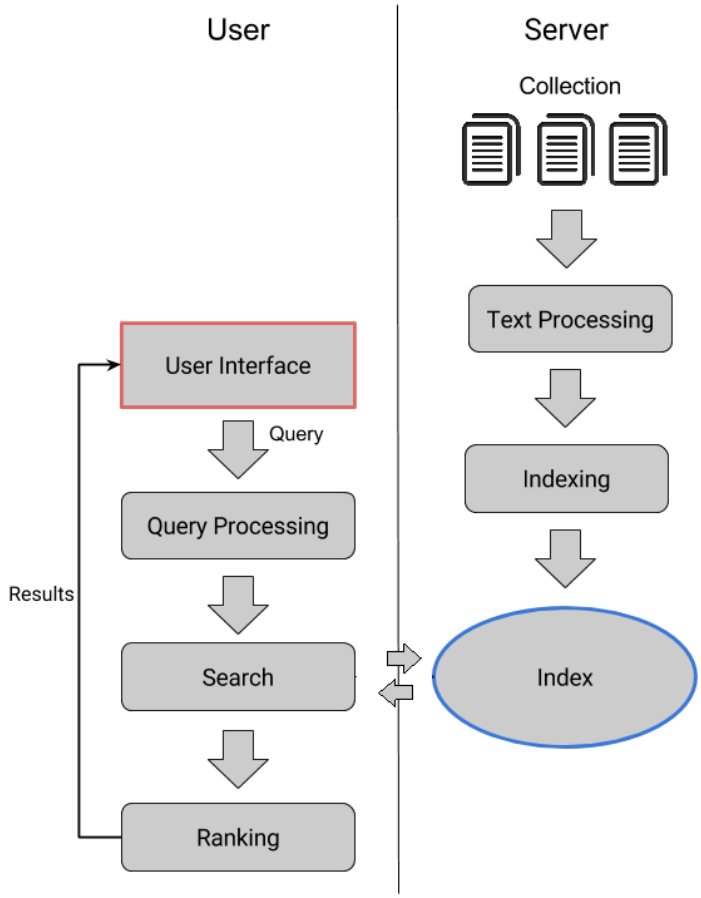

Figure 2.3: Generic Information Retrieval Process

2.2.1 INDEXING

The primary objective of indexing is to reduce the computational cost of searching a collection by enabling a user to query it without having to scan through it entirely. Data must be stored in an appropriate data structure, called indexes, before it can be efficiently queried. Indexes allow for faster and more efficient queries since they:

- Can store multiple fields, allowing data to be separated into different parts which can be independently represented, sorted and grouped according to different criteria. This allows for faster specific searches but consumes more disk space.
- Relations between entries and the collections can be set at the time of indexing (e.g., an entry for the word "dusk" can have a list of documents, where it occurs, associated with it).

- Can have established methods to determine the score of the results retrieved, based on a statistical analysis of the collection. This allows the system to rank query results according to their relevance to the query and the stored data.

There are many different types of architectures for an index, the most common for text retrieval being the inverted index. An inverted index consists of a dictionary, containing all the terms for a collection and a posting list that, for each term, contains every location in the collection it occurs, functioning as a pointer. This process results in a slower indexing time, however, it allows for a very efficient full search of the collection. This method has been around for a long time, yet it is considered so efficient for full-text search [28] that it is still used by the generality of the more powerful search engines used nowadays.

2.2.2 SEARCH ENGINES IN THEORY

A Search Engine is a IR system that implements the methods described above. It should provide an interface to the collection indexed, allowing the user to exploit the information stored without the need for thorough comprehension of its content. A search engine should be able to identify occurrences of a query and present them in a reasonable amount of time [29]. The architecture of a search engine is designed with two primary goals in mind [30]:

Effectiveness We want to be able to retrieve the most relevant set of documents possible for a query.

Efficiency We want to process queries from users as quickly as possible.

Unlike in the area of databases, where the dominating model is the relational model, search engine models are variable and their performance is highly related to the scope they are applied to. Some of the most relevant search engine models are described below

BOOLEAN RETRIEVAL

The first model of information retrieval ever conceived and one of the most used since its creation. It works as an exact-match retrieval because only documents that exactly match the query are returned. This type of model uses George Boole's mathematical logic to create sub-sets of documents from the stored collection [31].

Relevance, according to this model, is binary since a document either matches the query or not. This creates a problem for this model since, unless there is a post-processing step that involves the analysis of the retrieved documents, the solution does not provide any method to rank query results. The effectiveness of the model is very dependent on the user since he has the ability to create advanced queries utilizing operators from Boolean logic (**AND**, **OR** and **NOT**). This ends up being both the model's greatest strength and weakness [30][32] since it is entirelly dependent on the user's skill and knowledge.

The Vector Space Model was, historically, the first model to implement term weighting, ranking and relevance feedback [30][32]. This model assumes that documents and queries are part of a m-dimensional vector space, where m represents the vocabulary size, that is, the number of distinct terms in the collection. A collection composed of n documents can be represented as a matrix of term weights in which each cell describes the weight assigned to a term in a certain document:

$$\begin{array}{c|cccc} & \text{Doc}_1 & \text{Doc}_2 & \dots & \text{Doc}_n \\ \text{Term}_1 & p_{1,1} & p_{1,2} & \dots & p_{1,n} \\ \text{Term}_2 & p_{2,1} & p_{2,2} & \dots & p_{2,n} \\ \vdots & \vdots & \vdots & & \vdots \\ \text{Term}_m & p_{m,1} & p_{m,2} & \dots & p_{m,n} \end{array}$$

Table 2.1 shows an example regarding a collection composed of news titles related to the artist Tom Waits. In this example, the weight of each term is measured by its frequency in the sentence. Queries are also identified by a vector space. The vector of query "tom waits single" can be represented as Q=(0,0,0,0,0,0,0,1,0,1,1).

D_1 Tom Waits Album Review
D_2 Adele copies Tom Waits' Martha in Adele's new song
D_3 Listen to Tom Waits' new single

Terms	Documents		
	D_1	D_2	D_3
adele	0	2	0
album	1	0	0
copies	0	1	0
martha	0	1	0
new	0	1	1
listen	0	0	1
review	1	0	0
single	0	0	1
song	0	1	0
tom	1	1	1
waits	1	1	1

Table 2.1: Vector Space model matrix example

Using the vector space model, documents are represented on an m-dimensional space defined by the vocabulary terms. When a user query is represented in this same space, its similarity to the documents can be approximated by some measure of proximity in this context, thus we can rank documents based on their distance. We can establish a direct relation between distance and similarity since the closer the points represented by the vectors are, the more similar the documents/queries they

represent are.

However, this measurement is not direct and several optimizations have been developed with the most successful of those being the *Cosine Correlation* [33], mathematically defined in equation 2.1. This equation will determine the cosine of the angle between two vectors, if the vectors are identical, the angle between them will be 0 and thus, the cosine of the angle will be 1.

$$score(d_i, q) = \frac{\sum_{j=1}^{m} d_{ij} * q_j}{\sqrt{\sum_{j=1}^{m} d_{ij}^2 * \sum_{j=1}^{m} q_j^2}} \tag{2.1}$$

The weighting of the terms is generally independent for each solution, since the vector space model is independent of the type of weighting chosen. For efficient and reliable results, the term-weighting approach should be solution-driven [34].

. Several techniques are used, with the most common one being term frequency–inverse document frequency (tf-idf) [35].

tf-idf works by determining the relative frequency of words in a specific document compared to the inverse proportion of that word across the entire collection. This calculation determines how relevant a given word is in a certain document. tf-idf is formally presented as

$$tf\text{-}idf(w, d) = f_{w,d} \times \log(|D| / f_{w,D}) \tag{2.2}$$

where:

- D represents a document collection;
- w is a given word;
- d is an individual document where $d \in D$;
- $f_{w,d}$ equals the number of occurrences of w in the document d;
- $|D|$ is the corpus size;
- $f_{w,D}$ equals the number of documents in which the word w appears.

tf-idf uses both term-frequency ($f_{w,d}$) and inverted document frequency ($\log(|D| / f_{w,D})$) to obtain a metric for word relevancy.

Probability theory was used in an attempt to formally define term weighting and so, in 1977, Stephen Roberson formulated the *Probability Ranking principal* [36], stating:

"If a reference retrieval system's response to each request is a ranking of the documents in the collections in order of decreasing the probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data has been made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data."

In a probabilistic model, for each query, a document can be classified as relevant (R=1) or non-relevant (R=0) [30]. Considering D as the content of the document, two basic assumptions must be made when designing a probabilistic model:

- The relevance of a document is not dependent of any other document in the collection.
- D = { w_1, ... , w_n }; $_w$ represents a word in a vocabulary of size n, this collection of words is the representation of a document. Each word must have a value associated with it, either 0 or 1, indicating it is absent or present in a document, respectively.
- All words are mutually independent.

This allows us to determine for any given query:

$$P(R=1|D) \overset{rank}{=} \frac{P(D|R=1)}{P(D|R=0)} = \frac{\prod_{i=1}^{n} P(D_i|R=1)}{\prod_{i=1}^{n} P(D_i|R=0)} \tag{2.3}$$

This describes a very simplistic probabilistic model. This model does not take into account that some words are related to others since word independence simplifies the mathematical process. However many approaches seek to make improvements to this basic model:

- Efficiency improvement (save computational effort)
  - Normalize empty documents (all query words are absent) in order to provide higher efficiency.
  - Nonquery words cancel out, so only words that belong to D and Q are taken into account.

There have been many solutions created utilizing probabilistic models, and so it would be an extensive work describing them all. The most common algorithms implemented for this type of approach are the binary independence model [37] and the BM25 Ranking algorithm [38].

LANGUAGE MODEL

Language models are built for each distinct document. It is a probabilistic model that determines the probability of occurrence of a term in a collection. It is often called topic language model [30] since the model creates a probability distribution over the words contained in a document.

For the retrieval part of the model, documents are ranked by the probability of being able to produce the query from the collection of words that compose a document [39]. The generic equation that translates this process is defined in equation 2.4.

$$P(T_1, T_2, \ldots | D) = \prod_i P(T_i | D) \tag{2.4}$$

This presents a problem if a document does not contain a word that is queried, since it there will be a probability of 0 to produce a query result. In order to avoid this a technique named *smoothing* is applied, it consists on applying to every word a probabilistic value of occurrence, taking into account a default value or a background language model [40], containing all the words in a certain language and an associated probability.

2.2.3 SEARCH ENGINES IN PRACTICE

Several solutions have been implemented with many of the most used ones being open-source. We will be talking about three of the most used open-source search engines:

Lucene

Lucene is a search engine library created in 1999 that is supported by the Apache Sofware Foundation. It provides an API for communication and has had many platforms such as Solr [41] and ElasticSearch [42] built on top of it. It is one of the most widely recognized search engines [43].

Sphinx

Sphinx is an open-source full-text search server released in 2001. It also provides an API for communication.

SearchDaimon ES

SearchDaimon ES is an open-source enterprise search engine used for full-text search. It differs from the previous two since its focus is enterprise use, offering the ability to index rich documents (e.g., Powerpoint, Portable Document Format (PDF)) and images.

Search engines are characterized not only by their performance but also by the features they implement [44]. Some main features provided by most search engines are described below and Table 2.2 illustrates if and how they are supported by the search engines described above.

Query type The type of queries the index is capable of handling.

Storage How the index is stored (usually a file structure or a database).

Stopwords Whether the indexer can use a list of words that are too frequent in order to discard them.

Input Filetype The type of files that the indexer is capable of parsing.

Stemming Whether the indexer is capable of reducing words to their base forms, reducing inflected or derived words.

Sort If the index has the ability to sort the results according to certain criteria.

Ranking If the engine is capable of attributing a score to a result, presenting the results according to their relevance to the query.

Incremental Index If the indexer is capable of adding a file to an existing index without the need of rebuilding the whole index.

Faceted Search Whether the index can provide results that are filtered according to several criteria or fields.

	Lucene	**Sphinx**	**SearchDaimon ES**
Query Type	Boolean, Phrase, Wildcard, Proximity, Range	Phrase, Boolean, Proximity	Boolean, Phrase
Storage	Inverted Index	Inverted Index or MySQL DB	Inverted Index
Stop Words	Yes	Yes, but not default	Yes
Input File Format	Plain Text, HTML, Rich-Text	Plain Text, HTML, SQL Databases	Plain Text, HTML, Rich-Text, Http Sources, Databases
Stemming	Yes	Yes	Yes
Sorting	Yes	Yes	Yes
Ranking	Yes, allows pluggable models	Yes, multiple options (BM25 or proximity)	Yes
Incremental Index	Yes	Yes, however, uses 2 indexes, main on disk, auxiliary gets incremented and its necessary to batch update the main index with the auxiliary	Yes
Faceted Search	Yes	Yes, but not by default	Yes

Table 2.2: Main feature comparison

Taking into consideration the scope of this dissertation, the SearchDaimon tool is excluded due to the fact that it is not optimized for text documents due to its enterprise search nature. Comparing Lucene and Sphinx we can see some similarities. Both are fast and scalable, Sphinx is used by Craiglist.org and many companies such as Apple, Instagram and Netflix use Lucene or one of the

platforms built on top of Lucene [45]. Both have an extensive documentation describing their API. Although Sphinx integrates very well with relational databases, the scope of this dissertation does not require a relational database.

Since the performance of both engines provide is similar, we need to look at the features they provide and Lucene ends up being the one that packs a wider variety of features. Not only is the out-of-the-box experience better when using Lucene, the community around Lucene development is very active. The amount of platforms built on top of it and their success in the IR area shows its a proven technology. Furthermore, out of the many extensions to Lucene (e.g., Solr, Elasticsearch and Nutch), we decided to use ElasticSearch.

2.2.4 ELASTICSEARCH

ElasticSearch is an open-source search engine, created in 2010, that allows indexing documents and performing full-text search on the stored data. As stated before, it was built on top of Apache Lucene. At the date of writing, ElasticSearch is the most used search engine in the world [46], being used by companies such as Facebook, Netflix, eBay, Cisco, Microsoft and Adobe.

BASIC CONCEPTS

To truly understand how ElasticSearch works, the definition of some basic concepts, in the context of the framework [47], is required:

Cluster

A cluster represents a collection of nodes, that holds all stored data and provides the ability to search across all nodes it contains. It is identified by a name and all nodes that share the same cluster name are associated among themselves. Each cluster has a master node.

Node

A node is a part of the cluster, it represents a single server that is capable of storing data and provide search capabilities. Each node has a name for identification purposes. Nodes have the capacity of discovering other nodes in the network and, upon finding other nodes that share the same cluster name, they automatically join the same cluster. There is no limit for how many nodes can reside inside a cluster.

Index

An index represents a collection of documents, usually with similar characteristics. It is identified by a name which represents a logical namespace and it is used to refer to the index in all operations that concern the documents it contains. There is no limit for how many indexes you can have in a single cluster.

Document

A document is the basic unit of information. It is expressed in JavaScript Object Notation (JSON), a language-independent data format. Unlike most databases, it is not necessary to setup a schema for the document type before indexing a document, however, it is recommended for optimization purposes to create a Mapping.

Type

A type is a logical namespace, it allows users to define a schema, including fields and the data type for those fields, for the documents that are to be indexed within that namespace. You can define as many types as needed.

Shards

In an attempt to maximize performance and distributing the load between the system, ElasticSearch allows an index to be subdivided into multiple shards. Each index has a number of shards, defined at the moment of its creation. Each shard is an Apache Lucene instance, basically, it is the worker unit of the system and it acts as an independent "index" that can belong to any node in the cluster and its movable between nodes in case of node failure.

This allows ElasticSearch to horizontally scale according to the volume of data and to distribute and parallelize operations between the shards. All of these mechanics are completely managed by the architecture of ElasticSearch, following definitions set on a configuration file.

Primary Shards

Primary shards are where the information is stored and an index can have one or more primary shards. Each primary shards can have zero or more replica shards associated with it. A document is always indexed first on the primary shard and only after it is indexed on its replicas.

Replica Shard

Replicas work as a copy of the primary shard. Replication is an important part of the ElasticSearch architecture because it provides high availability in case of shard/node failure since a replica shard holds all information from a primary shard and can replace if it fails. It also enables queries to be executed in parallel throughout all replicas, increasing search throughput.

ELASTICSEARCH AS A FRAMEWORK

One can establish a parallelism, although a bit rough, between a conventional relational database and ElasticSearch, as described below.

Relational DB	$\Rightarrow$	Databases	$\Rightarrow$	Tables	$\Rightarrow$	Rows	$\Rightarrow$	Colums
ElasticSearch	$\Rightarrow$	Indices	$\Rightarrow$	Types	$\Rightarrow$	Documents	$\Rightarrow$	Fields

However, ElasticSearch has a combination of features that make it stand out from his competition:

API Specification

ElasticSearch is a recent tool and so, accompanying the emergence of RESTful [48] Web API's, all interactions with the engine are made through the use of a REST API using a JSON-based Query Domain Specific Language (DSL).

Real-Time Data Analysis

Since its built on top of Lucene, it has an incremental index, allowing for Near Real Time (NRT), since the latency between the moment a document has completed being indexed and the moment it is searchable is very little (variable, but usually less than one second).

Ample query options

Once again taking advantage of being a Lucene extension, ElasticSearch allows a broad variety of query options, offering auto-complete and "did-you-mean?" suggestions, geolocation and a powerful Query DSL, allowing for complex queries to be made in a standardized manner.

Scalability and High-Availability

ElasticSearch is a distributed System, allowing it to have a cluster divided amongst various nodes, shards are divided between a large number of servers, enabling a very strong horizontal scaling and a fail-safe mechanism that makes replicas become primary shards in case of shard failure. The fact that shards are able to adapt like this and move between nodes makes sure that the information provided by an index is always available as long as at least one node is online.

Document-Oriented & Schemaless

ElasticSearch's Document-Oriented nature allows users not to specify a schema before indexing JSON objects, since it can automatically detect field names and types. However, schemas are needed if you want to have a standard format for the documents, in order to execute in-depth analysis over them. In this context, we map a concept type to a *mapping*, defining fields and field types.

The fact that ElasticSearch is Document-Oriented allows the storage of nested documents. Since the operations of creating, deleting and updating a single document are atomic to ElasticSearch, being able to store documents inside other documents and querying them is a plus for use cases were you have intrinsic relations between your data.

QUERY DIVERSITY

Probably one of the strongest suits of ElasticSearch is the complex queries users are able to create due to the rich API provided along with the Query DSL. Below are some examples of the broad possibilities of querying provided by the API.

- Pagination : The parameters *from* and *size* define the offset of the first result to fetch and the size the amount of documents to be returned, starting from that offset.

- Sorting : Allows sorting on specific fields. It is possible to sort based on arrays or multi-value fields with operations such as sum, average and median of the values using the *mode* parameter. Nested objects can also be sorted.
- Highlighting : Highlights results found on specified fields.
- Scroll : Returns a large amount of results in a single request utilizing a cursor to traverse the data.
- Aggregations : An aggregation builds analytic information over a set of documents. Aggregations can be of different types: *a)* Bucketing tries to associate a key and a criteria to a bucket, all documents that follow a certain criteria are assigned to the bucket that matches it. *b)* Metric aggregations compute metrics over a collection of documents. *c)* Pipeline aggregations allow the creation of complex aggregations, applying them in succession.

All this reasons led me to choose ElasticSearch as the indexing tool for this dissertation, since it provides a rich and broad feature-set, enabling us to take full advantage of its structure in order to minimize computational overhead for processing the information retrieved.

2.3 WORD ASSOCIATION

According to the Cambridge Dictionary of Statistics [49] and the Oxford Dictionary of Statistics [50], association is a general term used to describe the relationship between two variables. Two variables are associated if some of the variability of one can be accounted for by the other.

Additionally, according to the Cambridge Dictionary of Statistics [49] two events are said to be independent if knowing the outcome of one tells us nothing about the other. Formally, independence is defined based on the probabilities of both events. Two events (A and B) are said to be independent if:

$$P(A \cap B) = P(A) \times P(B) \tag{2.5}$$

where P(A) and P(B) are the probabilities of events A and B.

2.3.1 ASSOCIATION MEASURES

Several measures have been devised [51] in order to determine whether or not the occurrences of two different words in a text collection are associated. Some of the measures that have been created are:

- Estimation of joint and conditional bi-gram probabilities;
- Mutual information and derived measures;
- Statical tests of independence;
- Likelihood measures;
- Heuristic association measures and coefficients.

However, several other implementations have been used to determinate word association, including machine-learning techniques to identify word association through event recognition.

2.3.2 LIKELIHOOD RATIO

The likelihood ratio test was first proposed by Dunning [52] in 1993, it was used identify word association under the assumption that the words in text had a binomial distribution. It provides an alternative to check if two simple hypotheses based on distribution parameters.

Two distinct hypotheses are formulated, taking into account two words (w_1,w_2):

$$\text{HYP1: } P(w_2|w_1) = P(w_2|\neg w_1) \tag{2.6}$$

$$\text{HYP2: } P(w_2|w_1) \neq P(w_2|\neg w_1) \tag{2.7}$$

Equation 2.6 represents an hypotheses formulated assuming both words are independent and equation 2.7 represents an hypotheses formulated assuming both words are not independent.

Both of the conditionals are used in the likelihood function [53]: $L(P(w_2|w_1), P(w_2|\neg w_1), \theta)$, where θ represents the parameter of the binomial distribution $b(n, k, \theta)$.

Assuming:

- Under HYP1 described in Equation 2.6 consider:
  - $p_1 = P(w_2|w_1) = P(w_2|\neg w_1)$
- Under HYP2 described in Equation 2.7 consider:
  - $p_2 = P(w_2|w_1)$
  - $p_3 = P(w_2|\neg w_1)$

$$\lambda = \frac{L(P(w_2|w_1); p_1) \times L(P(w_2|\neg w_1), p_1)}{L(P(w_2|w_1); p_2) \times L(P(w_2|\neg w_1); p_3)} \tag{2.8}$$

Equation 2.8 represents the likelihood ratio.

2.3.3 Pointwise Mutual Information

The concept of Pointwise Mutual Information (PMI) [54] was defined by Robert Fano [55], and states that if two points (in the context of this dissertation: words), x and y, have probabilities P(x) and P(y) and a joint probability P(x, y), then their mutual information, MI(x, y) , is defined to be

$$\text{MI}(x, y) = \log(\frac{P(x, y)}{P(x)P(y)}) \tag{2.9}$$

PMI compares the probability of observing events x and y together (their joint probability) with the probabilities of observing x and y independently (chance). If there is a genuine association between x and y, then their joint probability P(x, y) will be much larger than chance P(x)P(y) and so MI(x, y) > 0. However, if no interesting relationship is found P(x, y) $\approx$ P(x)P(y) then MI(x, y) ≈ 0. If x and y

are in complementary distribution, then P(x, y) will be much smaller than P(x)P(y), making MI(x, y) < 0.

Algorithm 1: Computing PMI

Input : word pair (x,y)
Output : PMI for word pair (x,y)

let $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ // For a collection of n documents;
N = length of $\mathcal{D}$;
(df_x, df_y) = document frequencies of x and y;
$(p(x), p(y)) = (df_x/N, df_y/N)$;
$co\_occurences = 0$;
for $i \leftarrow 1$ ***to*** N **do**
    if (x, y) ***in*** D_i **then**
        $co\_occurences+ = 1$;
    end
end
$p(x, y) = co\_occurences/N$;
$PMI = \log \frac{p(x,y)}{p(x)p(y)}$;

2.3.4 PEARSON'S χ^2 TEST

Pearson's chi-squared test [56], as a test of independence, evaluates whether odd observations on two variables are independent from each other. These variables are categorical and can be represented in a contingency table.

Occurrences	X+	X-
Y+	P(X∩Y)	P(Y|¬X)
Y-	P(X|¬Y)	P(¬X) ∩ P(¬Y)

Table 2.3: Contingency Table example for Word Association

Table 2.3 represents two events X and Y, stating the document count in which they appear, the plus sign (+) represents the occurrence of the concept in the document and the minus sign (−) means the concept is not present in the document.

The chi-square test of independence begins with the hypothesis of no association between the two variables. This creates a complementary hypothesis that states that two variables are somehow related. Assuming the two variables are X and Y, then

$$\begin{aligned} &\text{HYP0: There is no relationship between } X \text{ and } Y \\ &\text{HYP1: There is some relationship between } X \text{ and } Y \end{aligned} \tag{2.10}$$

The Pearson's chi-squared statistic is obtained after considering the following equations

$$E_{x,y} = No_x o_y \tag{2.11}$$

where $E_{x,y}$ represents the theoretical frequency for each cell and N is the sum of all cells in the table.

$$o_x = \frac{O_x}{N} = \sum_{y=1}^{c} \frac{O_{x,y}}{N}$$
$$o_y = \frac{O_y}{N} = \sum_{x=1}^{r} \frac{O_{x,y}}{N} \tag{2.12}$$

where:

- O_x is the number of observations of type x
- O_y is the number of observations of type y
- r is the number of rows
- c is the number of columns

The χ^2 statistic is given by

$$\chi^2 = \sum_{x}^{r} \sum_{y}^{c} \frac{(O_{x,y} - E_{x,y})^2}{E_{x,y}} \tag{2.13}$$

Taking into account equations 2.11 and 2.12, equation 2.13 can be resolved into

$$\chi^2 = N \sum_{x,y} o_x o_y \left(\frac{(O_{x,y}/N) - o_x o_y}{o_x o_y} \right)^2 \tag{2.14}$$

A chi-squared probability of less than or equal than a defined value (usually 0.05) is sufficient to reject the hypotheses that X is independent from Y.

2.4 DATA VISUALIZATION

One of the most important aspects of a IR framework is having the ability to meet the information needs of the user. Keeping that in mind, it is safe to say that the data must be presented in such a manner that the information needs are met explicitly, preferably without having to resort to arduous customization of the interface.

2.4.1 EFFECTIVE DATA VISUALIZATION

Visualization tools in the field of IR should attempt to visually stimulate the user and its ability to identify patterns, in other words these tools should enable cognition [57]. In 1993, Norman [58] proposed some basic principles for developing efficient representations:

Appropriateness Principle

The visual representation should provide only the exact amount of information that is needed for the task it must complete. Additional information can be distracting and makes the cognitive

process more difficult.

Naturalness Principle

Experimental cognition is more effective when the aspects of the visual representation are closer matches to the information being represented. This principal supports that when a representation does not match the cognitive model of the information established by the user it may hinder his understanding.

Matching Principle

Effective visual representations should be suggestive of the action they execute. Representations are more effective when they match the task to be performed.

In 2002, Tversky [59] suggested the following two principles:

Principle of Congruence

The structure and content of a visual representation should be directly related to the structure and content of the desired mental representation. What is represented should represent the important concepts to retain in the domain of interest.

Principle of Apprehension

The structure and content of a visual representation should be straight-forward to perceive and comprehend, in terms of velocity and accuracy.

2.4.2 DATA VISUALIZATION ADVANTAGES

There are advantages in presenting complex data in a form other than a text-oriented approach [57], such as:

Clarity

The user is able to understand more quickly the content of a visual representations than an aggregation of numbers.

Relativity and Proximity

Capacity to show relative sizes, similarities and differences of groupings in a straight-forward way.

Concision

Ability to present large amounts of data, from different contexts, at the same time.

Context

Ability to highlight part of the visualization while still being able to see how the highlighted part is positioned in its relational context.

"Right Brain" stimulation

Information displayed stimulates the user to utilize intuitive, reactive or spatially oriented cognitive processes in order to identify patterns.

2.4.3 DATA VISUALIZATION APPROACHES

In 1983, Edward Tufte [60] describes his idea of what the aim of graphical displays should be:

"Excellence in statistical graphics consists of complex ideas communicated with clarity, precision and efficiency. Graphical displays should:

- show the data;
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production or something else;
- avoid distorting what the data has to say;
- present many numbers in a small space;
- make large data sets coherent;
- encourage the eye to compare different pieces of data;
- reveal the data at several levels of detail, from a broad overview to the fine structure;
- serve a reasonably clear purpose: description, exploration, tabulation or decoration;
- be closely integrated with the statistical and verbal descriptions of a data set.

Graphics reveal data. Indeed graphics can be more precise and revealing than conventional statistical computations."

Since then several types of graphic display have been developed to explicitly represent different types of information in a fashion that will better satisfy the information needs of the person visualizing them. Some such displays are described below.

CONCEPT GRAPH VISUALIZATION

The idea of a conceptual graph was first introduced by John Sowa to represent the conceptual data model used in a database [61]. Concept graphs provide a basic visualization tool representing a tree where concepts are the tree's nodes.

The most simple concept graph visualization is a hierarchical tree as illustrated in Figure 2.4.

Association graphs, although based on the previous approach, are used to evidence relations between concepts. Each vertex of the graph has a concept and concepts are connected by edges if the association is relevant according to a preset threshold.

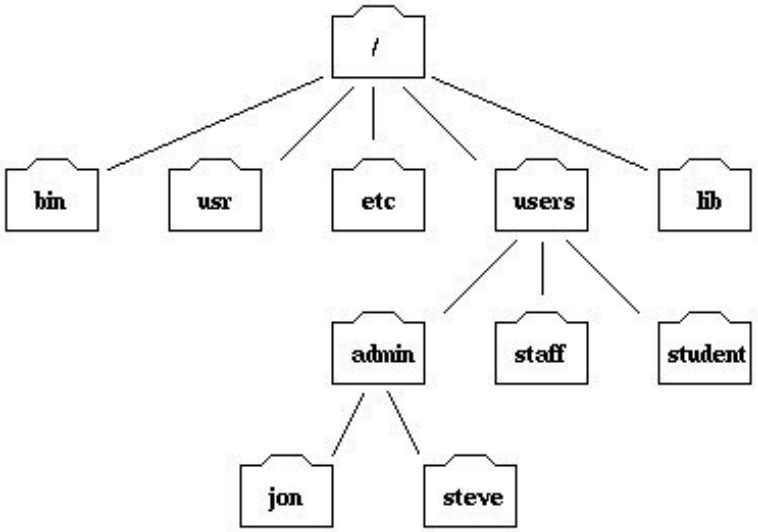

Figure 2.4: Hierarchical representation of a partial file system

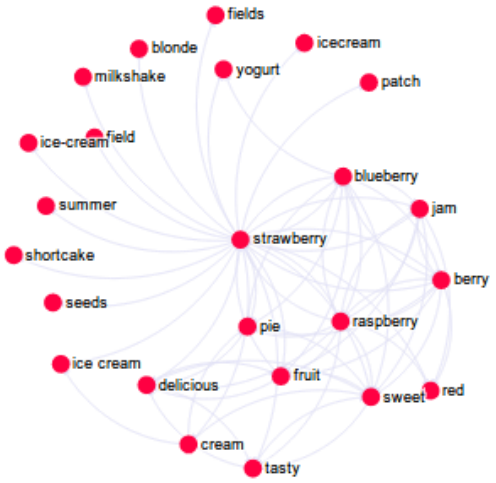

Figure 2.5: Association graph example

Association graphs provide a simple way to infer conclusions about data that would otherwise require thorough data exploration. It also allows users to find associations that are so subtle that without resorting to a similar tool would be almost impossible to identify. Figure 2.5 displays such a graph.

HISTOGRAMS

The concept of presenting the probability distribution of a continuous variable was introduced by Karl Pearson [62]. Histograms display data by grouping the data into a set of intervals (bins) and

counting the frequency of the values in each interval. This makes it a very useful tool to represent continuous data due to the fact that is easier to identify tendencies in data progression. However, since data is divided among categories, it lacks the precision to analyze exact values. Figure 2.6 displays a generic histogram.

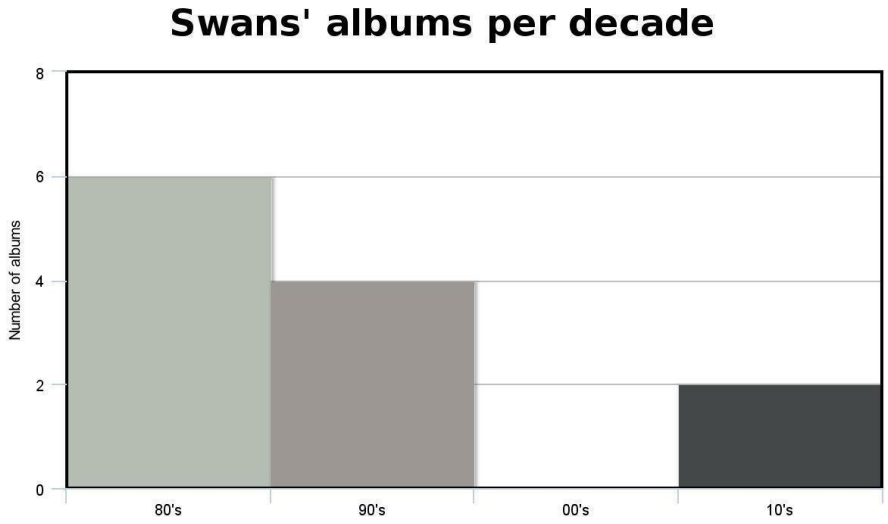

Figure 2.6: Histogram displaying the number of Swans' albums (frequency) over the span of decades (categories)

CIRCLE GRAPHS

Circle Graphs are an approach that mainly tries to evidence relationships between nodes. They excel at showing large amounts of data given a query set [63], therefore, they are extremely useful in the visual representation of associations. They are especially valuable when used as a complement to other graphical representations and even the simultaneous display of multiple circle graphs can help users to infer direct and indirect comparisons between different query results.

As displayed in Figure 2.7, items are mapped around the circumference and relations between items are represented by edges that intertwine the items across the interior area of the circle.

An example of the application of this representation is Circos. Circos is a visualization tool created to facilitate the identification and analysis of similarities and differences arising from comparisons of genomes [64].

Circle graphs can be visually further enhanced by using color to identify different types associations, directionality or to group items. Line thickness can also be used to determine frequency, importance or strength of the several associations.

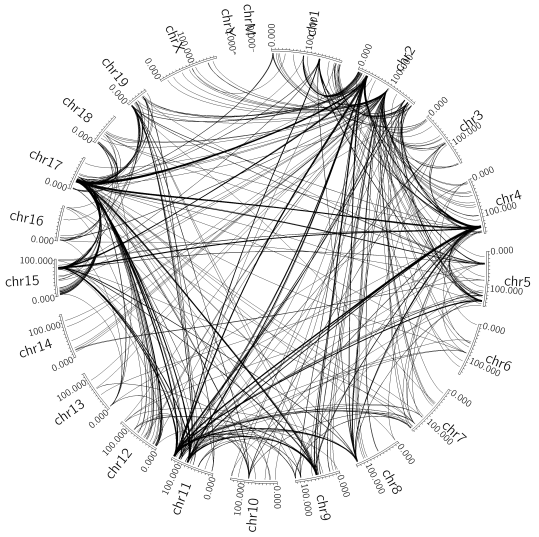

Figure 2.7: Circle Graph using Circos' software for Interchromosomal Internal

TREEMAP

The Treemap representation was initially proposed to overcome the challenge of displaying large hierarchical structures, such as directory structures in hard disk drives. It was proposed by Ben Shneiderman [65].

The Treemap representation follows a couple of rules: *a)* Data should occupy all available space on the screen. *b)* Data is to be grouped in a geometric form (usually rectangles), with the possibility of nesting forms inside others. It utilizes a recursive algorithm to divide the data, allowing this representation to be very efficient in representing thousands of nodes.

A tiling algorithm must be defined, it is responsible for the placement and dimensioning rectangles. Usually there is a trade-off in the process of placing and ordering query results, since prioritizing one of this process causes the other to become more unpredictable [66].

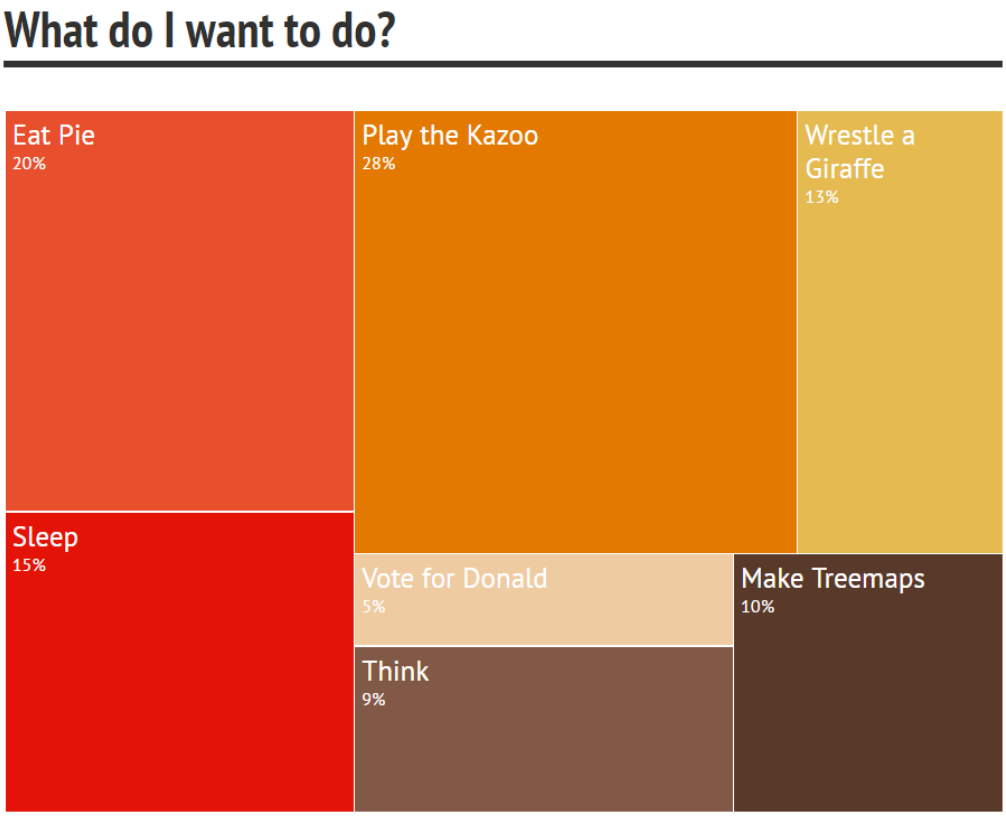

Figure 2.8: Treemap Example

2.5 RELATED WORK

2.5.1 FACTA+

Facta+ [67] is a real-time text mining system for finding and visualizing direct and indirect associations between biomedical concepts based on MEDLINE abstracts. The system can be used as a search engine for MEDLINE articles or as a visualizer for concept associations, covering biomedical fields like genes, diseases and chemicals.

The system uses a machine-learning based approach to detect events in text, based on NLP, using verbs as a mean to establish relationships between biomedical concepts, estabilishing all direct associations this way. To discover indirect associations Facta combines known associations, i.e if concept A is directly associated with concept B, and concept B is directly associated with concept C, a potential connection between concepts A and C is hyphotesized if theres is not a direct association between A and C already. This is called Swanson's ABC Model [68].

Facta+ Visualizer [67] uses a Treemap to display biomedical concepts that co-occur with, and relate to, a biomedical concept that is declared in a query. Larger rectangles are divided by color, each one representing a different semantic group and each smaller rectangle represents a biomedical concept that belongs to that group. The higher the score a relation has, the larger the area that a rectangle occupies on screen.

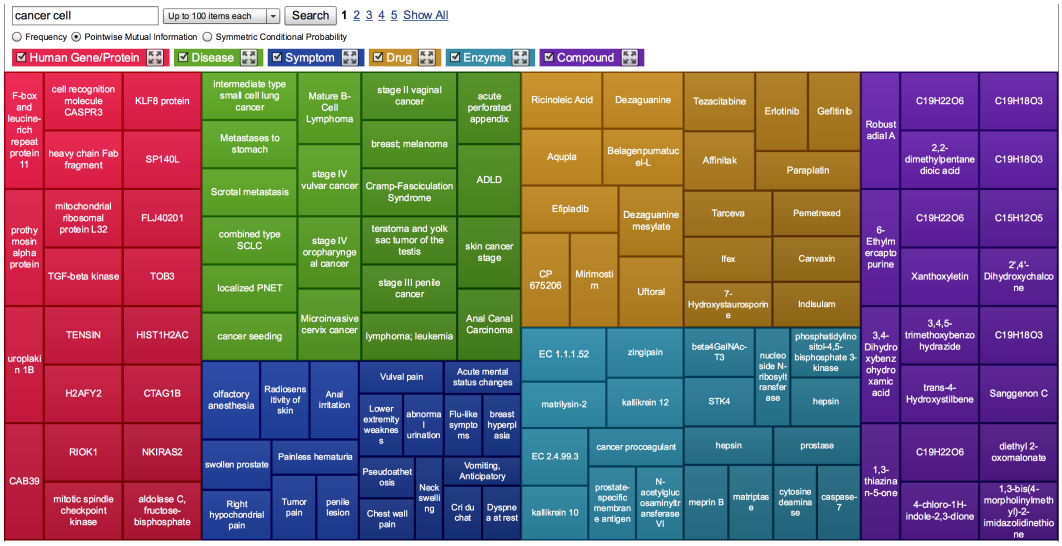

Figure 2.9: Facta+ Visualizer's Treemap

2.5.2 DIGSEE

Digsee [69] is an application developed with the objective of querying MEDLINE abstracts in search for sentences that evidence the association between diseases and genes that are related to those diseases.

Digsee uses a graph-based approach that extract events between genes and proteins [70] and extends it by applying a text-mining method that further identifies relations of extracted events between genes and diseases.

Digsee's interface allows for two distinct functions:

- Use Digsee as a search engine to service evidence sentences that associate queried genes with a specific disease defined in the query.
- Create and display a graph of gene co-occurences in the context of a specific query.

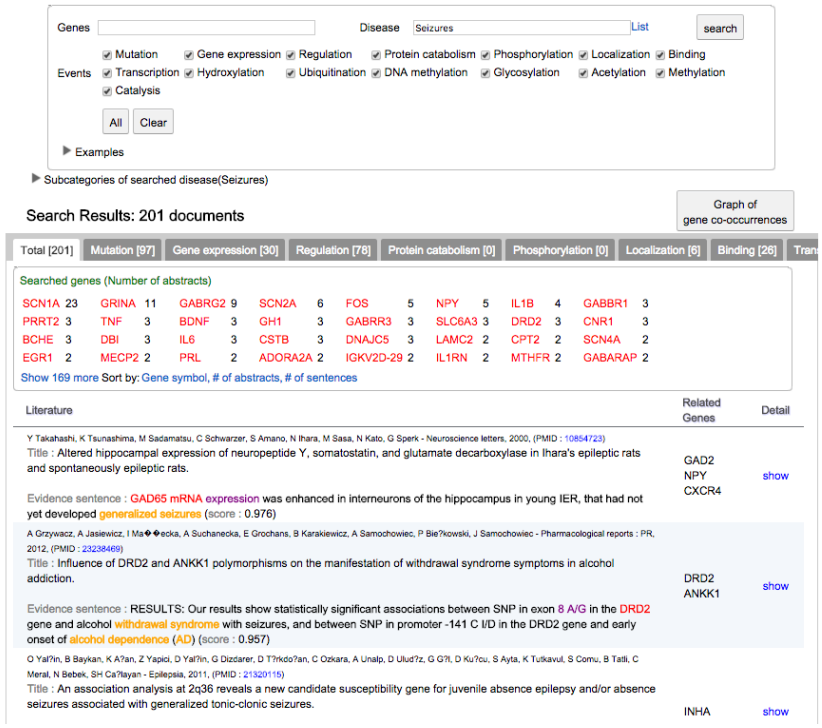

Figure 2.10: Digsee as a search engine

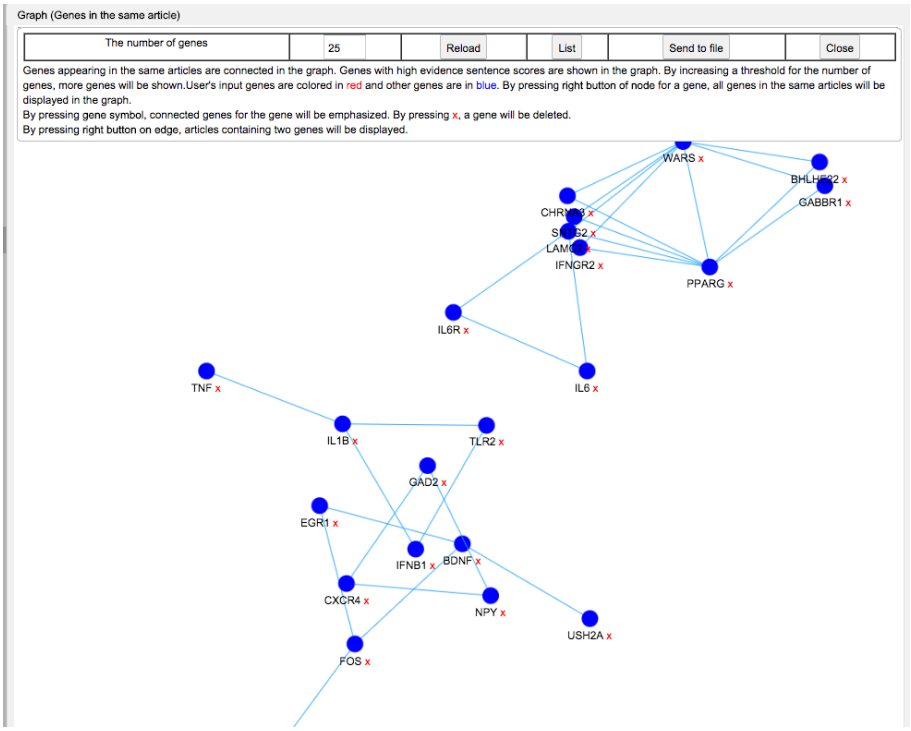

Figure 2.11: Digsee's gene co-occurrence graph

2.5.3 KNOWLEDGE.BIO

Knowledge.Bio [71] is a web platform that allows users to, using a graphical user interface, build and evaluate maps of concepts and their relationships. Conceptual relations are extracted from the Semantic Medline Database (SemMedBD) [72] and from Implicitome [73], two resources originated from text mining of MEDLINE abstracts.

Semantic relations are retrieved from the SemMedBD, which holds more than 70 million relations extracted from MEDLINE abstracts using the SemRep NLP system [74]. Gene-Disease associations are integrated from the "Implicitome".

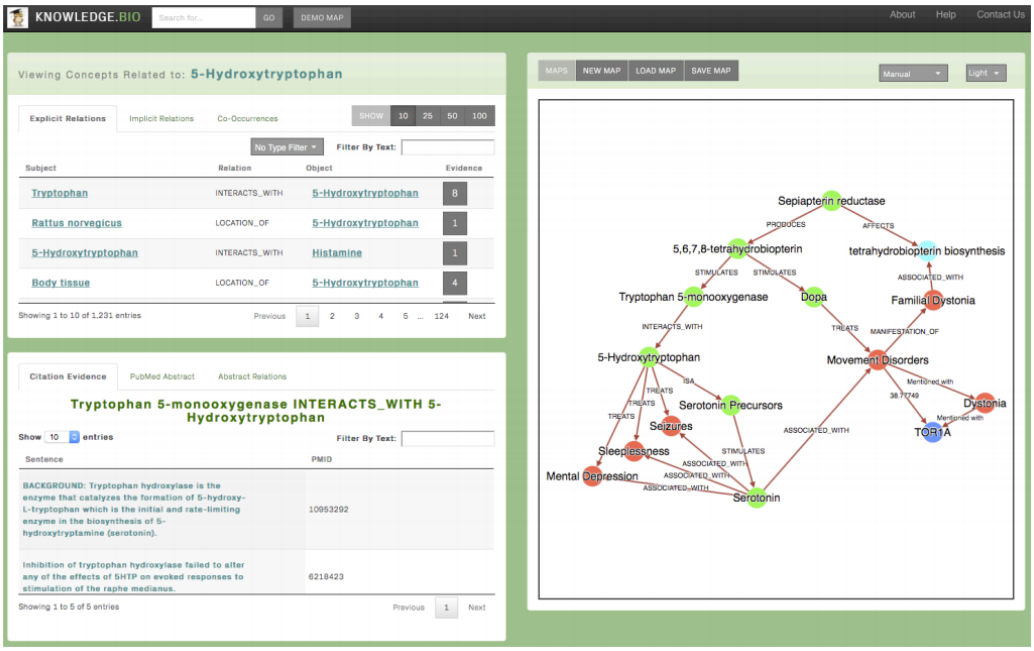

Figure 2.12: Knowledge.bio User Interface

As seen in Figure 2.12, the web application allows the user to browse through this concept space, identifying relations of interest, browsing supporting MEDLINE articles and constructing a graph of relationships, where the type of relationship between concepts is displayed.

CHAPTER 3

Architecture and Implementation

This section provides an in-depth explanation of the system's architecture and implementation, providing a section for each main component of the project's pipeline and explaining decisions and compromises made throughout the applications' design.

Since the main focus of this dissertation is the task of efficiently storing and presenting annotated MEDLINE articles, there is a need to implement a pipeline that allows us to transform semi-structured data into a fully explicit, concept-driven, graph visualization. The project's architecture is described in figure 3.1.

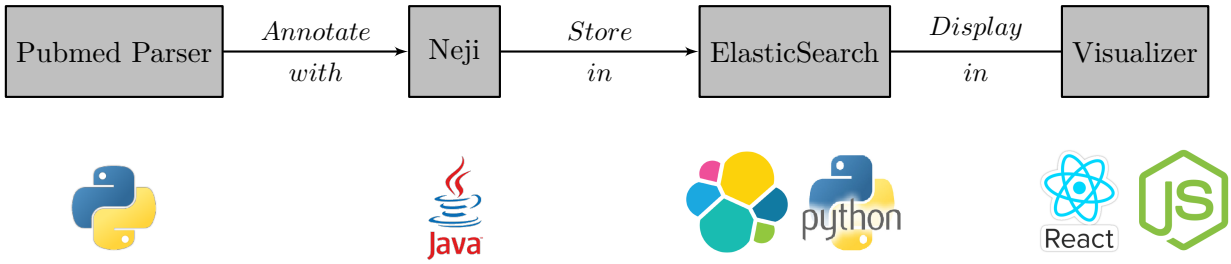

Figure 3.1: Data Processing Pipeline

The overall pipeline's structure is as follows:

- First, the Pubmed Articles are retrieved using the National Center for Biotechnology Information (NCBI)'s API. PubMed Articles are made available in the XML format, therefore an appropriated reader is required to parse them. Attributes such as the abstract, title, journal name, authors, year of publication and Medical Subject Headings (MESH) ID's are identified, as we crawl the articles, and are extracted;

- As documents are parsed, we use Neji's WebServices, a REST API that integrates the NEJI framework into a RESTful Server. The documents' title and abstract are sent to the server for annotation purposes and the server returns the annotated text in the JSON format, ready to be indexed into ElasticSearch. This format will be explained in detail posteriorly;
- When all the information regarding a document is parsed and retrieved, it is indexed into ElasticSearch using the Python driver for ElasticSearch. A log is kept on a separate index for the purpose of evaluating average indexing time;
- A Node.js Server is deployed on the same machine. This server is conntected to ElasticSearch and establishes the bridge between our data visualization platform and the data storage. The Node server implements a simple REST API, as well as a set of WebSocket listeners for the Front-End Application;
- The last piece of our pipeline is the web application provided, being the most important part of the project, as far as the user is concerned. It is implemented in React.js, a Framework developed by Facebook. A visualization module was also built, from scratch, using the d3.js visualization library. Thanks to it we are able to display, in the form of a graph, data stored in our Search Engine in real-time.

The architecture for the Web Server is described in figure 3.2.

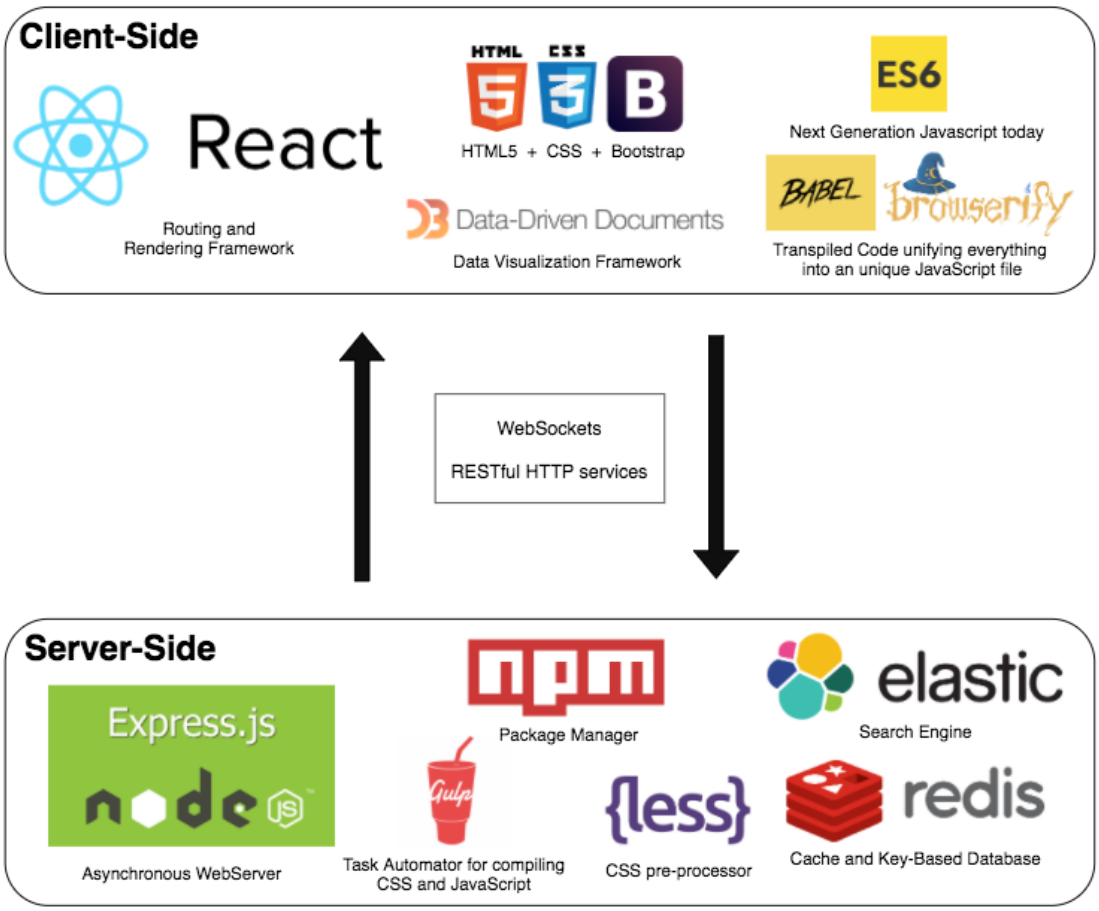

Figure 3.2: Web Server architecture (Front-end and Back-end)

Although, at first sight, the number of features and frameworks may seem overwhelming, every element has a part to play and most are used as a mean of optimization, whether it is to increase the system's performance or simply to streamline and accelerate the development process.

This structure can be roughly divided into four different modules:

- Annotation
- Storage
- Back-end
- Front-end

During the following sections, an in-depth approach to each individual module will be described, evidencing the choices that were taken, compromises that were made and problems that had to be overcome as we progressed through the development of this project.

3.1 ANNOTATION

The tasks of NER and WSD are handled by the NEJI framework, implemented in Java, through dictionary-based recognition with the normalization process being executed using a set of dictionaries. Text Segmentation methods like sentence splitting, tokenization, POS tagging, lemmatization and chunking are provided by Neji's customized version of the GDEP parser [75].

The set of chosen dictionaries plays a very important part in this project since they are the sole source of knowledge for the tasks of NER, WSD and normalization, so it was important to guarantee their reliability. The set used was based on sets that had previously been proven to be successful, as published by Campos et al. [12] and by Nunes et el. [76].

The following biomedical concepts type are taken into consideration: disorders, cells, cellular components, biological processes and molecular function using UMLS [13] as the data source, chemicals using extension of the ChEBI [77] and JoChem databases and genes and protein using LexEBI [78] as a data source.

In the context of this dissertation, we can treat the annotations provided by Neji as a simple hierarchical structure:

- The text to be annotated is, in the first place, parsed and split in different sentences. Each sentence has an identifier that allows us to determine its order in the text and two integers, one representing the starting position of the sentence and the other the final position.
- Each sentence has a list of annotations, each annotation is described by a starting and final position, a list of Concept Unique Identifier (CUI)s and the semantic group of the annotated concepts.

Neji's annotation results can have different formats, however, a special writer was implemented for GRACE, it is in the JSON format and is described in Listing 1 and Listing 2:

```
// TEXT
{
    "text":[ /*SENTENCE ARRAY*/ ]
}

// SENTENCE
"sentence":{
    "id":{"type": "integer"}, // Sentence ID
    "begin":{"type": "integer"}, // Sentence Begin
    "end": {"type": "integer"}, // Sentence End
    "annotations": [ /* ANNOTATIONS ARRAY */ ]
}

// ANNOTATIONS
"sentences":{
    "group":{"type": "string"}, // Array of semantic groups for
       annotation
    "begin":{"type": "integer"}, // Annotation Begin
    "end": {"type": "integer"}, // Annotation End
    "cui": {"type": "string"} // CUI Array
}
```

Listing 1: Output Layout

```
{
  {
  "text": [{
    "sentence": {
      "begin": "0",
      "end": "362",
      "id": 0,
      "occurences": [{
        "group": "CHEM",
        "begin": 22,
        "end": 34,
        "cui": ["C0000966"]
      }, {
        "group": "PROC",
        "begin": 35,
        "end": 44,
        "cui": ["C0030011"]
      }]
    }
```

```
  }, {
    "sentence": {
      "begin": "363",
      "end": "521",
      "id": 1,
      "occurences": [{
          "group": "CHEM",
          "begin": 379,
          "end": 405,
          "cui": ["C0050417"]
        },
        {
          "group": "PRGE",
          "begin": 445,
          "end": 457,
          "cui": ["C1426584"]
        }, {
          "group": "PROC",
          "begin": 445,
          "end": 457,
          "cui": ["C1153524"]
        }
      ]
    }
  }]
}
}
```

Listing 2: Output Example

3.2 STORAGE

When it comes to storing data, multiple solutions are available with the increasing concern in the technological world for efficient and reliable storage processes (whether it is hardware or software). This is a key part of the implementation described in this dissertation and one that required the study of many different approaches.

3.2.1 THE DATA

The data we will be storing is several MEDLINE articles, each of them can be described by the following components, roughly based on the information provided by National Library of Medicine

(NLM) (https://www.nlm.nih.gov/bsd/mms/medlineelements.html). (Fields marked with an * are stored in our system, however they are not used in the web application):

Title
The article's title as displayed in PubMed, this field will be parsed and annotated by the Neji Framework and the storage of that information is also a requisite.

Abstract
The article's abstract as displayed in PubMed, this field will also be parsed and annotated by the Neji Framework and the storage of that information is also a requisite.

Authors
The list of authors that contributed to the article, as presented in PubMed.

MESH *
The list of MESH, used to characterize the content of the article.

Language *
The language in which the article was published; in the context of this dissertation only English articles are taken into account.

PMID
A unique identifier for each PubMed Article. This is used to provide direct links to articles in PubMed.

Publication Year *
The year that the referred article was published.

Journal Title
This field contains the full journal title, taken from the NLM cataloging data and following its rules for how to compile a serial name.

Journal Title Abbreviation *
The abbreviation for the journal title.

Article Identifier *
This field is populated by the publisher. It contains a identifier that links with the publisher's record system. It may contain a Publisher Item Identifier (PII) or an Digital Object Identifier (DOI). This information is very important so we can provide a direct link to an article's full text, if available.

Although the MEDLINE API provides several other fields, their importance was judged not to be relevant, for our research's purpose. As described above, the title and abstract fields will be parsed and annotated, therefore there is a need for correctly storing that information since the core of this dissertation consists on efficiently querying those annotations.

3.2.2 REQUIREMENTS

Efficient storage is a requisite of this project, since not only is the sheer volume of data a huge predicament to deal with, but there is also a requirement for accuracy and efficiency in retrieving

documents. Taking into account both these premises we can easily point out that scalability is an issue, as well as the importance of a well defined structure, providing a correct semantic representation of the information described in the previous sections. Given the snippet showing in Listing 1, there is a obvious need that the database system must allow for complex data structures such as nested objects.

Taking into account the premises defined above, there was a need to define what was the type of storage that best fit our data needs.

Analyzing the available options of database storage, the two major types of databases are SQL and NoSQL. Table 3.1 illustrates some differences between characteristics of SQL and NoSQL databases [79]:

SQL	NoSQL
SQL Databases have a specific schema.	NoSQL Databases can have a dynamic schema to account for unstructured data.
SQL Databases follow a relational model, composed by rows and columns. Rows contain the information regarding each entry and each represents a specific data field for that entry.	NoSQL can have many different data storage models. The most important are document-based, graph-based, columnar and key-value.
SQL databases tend to scale vertically, this means that as data grows, the server needs to be bigger. This is an expensive process and although there are distributed SQL solutions, it is not the norm	NoSQL databases scale horizontally, this means that data storage and processing is spread across multiple servers, this makes for a more cost-effective solution than vertical scaling.
SQL databases have an underwhelming performance with hierarchical data	NoSQL databases with key-value pair proprieties tend to be better fits for this type of data

Table 3.1: SQL and NoSQL differences

Another important concept to take into account is Atomicity, Consistency, Isolation, Durability (ACID) [80], which is a set of properties that were established with the aim that Database Management System (DBMS) models would guarantee that transactions were successfully and concurrently processed. SQL databases follow very rigidly this model, avoiding data loss at all costs whereas NoSQL databases tend to sacrifice ACID compliance for performance.

Taking the above arguments and our data model into account it was obvious that a NoSQL database would be the best fit for this implementation. After analyzing different models, a document-based

model proved to be the one that was the closest to our data representation. However, since articles are text documents and full-text search and indexing are valuable additions to the implemented system, ElasticSearch was chosen as the framework of choice due to the nature of its implementation and the complex query methods it possesses, namely aggregations.

3.2.3 ELASTICSEARCH

In order to have a successful deployment environment it is necessary to study the framework's requirements and specifications. ElasticSearch runs in a Java Virtual Machine (JVM) environment, therefore it is highly adaptable and runs on most systems, each instance/node is configured by two different means:

Configuration File
The configuration file defaults to the name "elasticsearch.yml", most configurations are defined in this file.

Environment Variables
ElasticSearch also uses Environment Variables for configuration, if they are set up.

This project's configuration consists of two distinct configuration files and the definition of Environment Variables, as follows:

```
1  cluster.name: grace
2
3  node.name: "grace_master"
4
5  node.master: true
6  node.data: true
7
8  network.host: localhost
9
10 indices.fielddata.cache.size: 75%
11 indices.breaker.fielddata.limit: 85%
12
13 bootstrap.mlockall: true
```

Listing 3: elasticsearch1.yml

```
1  cluster.name: grace
2
3  node.name: "grace_slave"
4
5  node.master: false
```

```
6 node.data: true
7
8 // Same as configuration file above, starting from this line
```

Listing 4: elasticsearch2.yml

A brief explanation of the configuration represented above, concepts marked with an * are not explicitly defined but their understanding is relevant:

cluster.name
: The name for the logical namespace that aggregates a collection of nodes. In this case both nodes share the same cluster.

node.name
: The name of the individual node.

node.master
: Whether the node is allowed (or not) to be a master node.

network.host
: The IP address or hostname to which the node will bind itself to.

Field Data *
: Although not explicitly configured, it is an important concept to understand. The field data cache loads field values into memory, allowing for faster data access. However this is an expensive process, so it is recommended to have a fair amount of memory available so the data can be kept in memory.

indices.fielddata.cache.size
: The setting defines the percentage of the heap size reserved for the field data cache, since the deployment environment has a fair amount of resources available the percentage allocated is high so ElasticSearch is capable of keeping a higher volume of data in cache.

Circuit Breaker *
: The Field Data cache presents a problem, since its size is only checked after the data is loaded into memory, if the loaded data exceeds the available memory an OutOfMemoryException occurs. In order to avoid this, the circuit breaker attempts to estimate the memory requirements of each query, taking into account the fields involved and their characteristics. Since this calculation is done *before* the query, if the data size exceeds the available memory the query will be aborted and an OutOfMemoryException won't be raised.

indices.breaker.fielddata.limit
: The field data circuit breaker will limit queries with a size above the percentage of the heap space defined in this field.

bootstrap.mlockall
: ElasticSearch loses a lot of performance if its process is swapped out and swap memory is used, therefore if this flag is set to true ElasticSearch's memory will be locked into RAM.

The only environment variable defined for this implementation is ES_HEAP_SIZE, which defines the heap size allocated for the JVM that runs ElasticSearch. It is configured by running the following command (in Unix):

```
export ES_HEAP_SIZE=10g
```

Listing 5: ElasticSearch Environment Variables

Our implementation takes advantage of the key-value proprieties of our data model, each article's attribute can be considered a key and its contents the value. For instance:

```
{
    PMID: 26039902,
    Title: "The Pain Course: a randomised controlled trial examining
       an internet-delivered pain management program when provided
       with different levels of clinician support"
}
```

Listing 6: Simple Article JSON

Although ElasticSearch supports schemaless indexing, it isn't the most reliable approach to a problem like this, since not only the data structure is known beforehand but also there is a desire for queries to be optimized, therefore, the definition of an index schema seems to be a mandatory step in the process. Our database schema is defined in Listing 7 and is explained in-depth below.

```
{
  "publication": {
    "properties": {
      "mesh": {
        "type": "string", "index": "not_analyzed", "doc_values":
           "true"
      },
      "abstract": {
        "type": "string", "index": "analyzed"
      },
      "authors": {
        "type": "string", "index": "not_analyzed", "doc_values":
           "true"
      },
      "doi": {
        "type": "string", "index": "no"
      },
      "jrn_title": {
```

```
17          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
18        },
19        "jrn_title_abbr": {
20          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
21        },
22        "lang": {
23          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
24        },
25        "pmcid": {
26          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
27        },
28        "pmid": {
29          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
30        },
31        "title": {
32          "type": "string", "index": "analyzed"
33        },
34        "cui_lst": {
35          "type": "string", "index": "not_analyzed", "doc_values":
                "true"
36        },
37        "publ_year": {
38          "type": "integer", "index": "not_analyzed", "doc_values":
                "true"
39        },
40        "abstract_ann": {
41          "type": "object",
42          "properties": {
43            "sentence": {
44              "type": "object",
45              "properties": {
46                "id": {
47                  "type": "integer", "index": "not_analyzed",
                        "doc_values": "true"
48                },
49                "begin": {
50                  "type": "integer", "index": "not_analyzed",
                        "doc_values": "true"
51                },
52                "end": {
```

```
                "type": "integer", "index": "not_analyzed",
                    "doc_values": "true"
              },
              "occurrences": {
                "type": "object",
                "properties": {
                  "group": {
                    "type": "string", "index": "not_analyzed",
                        "doc_values": "true"
                  },
                  "begin": {
                    "type": "integer", "index": "not_analyzed",
                        "doc_values": "true"
                  },
                  "end": {
                    "type": "integer", "index": "not_analyzed",
                        "doc_values": "true"
                  },
                  "cui": {
                    "type": "string", "index": "not_analyzed",
                        "doc_values": "true"
                  }
                }
              }
            }
          }
        },
        "title_ann": {
          "type": "object",
          "properties": {
            "sentence": {
              "type": "object",
              "properties": {
                "id": {
                  "type": "integer", "index": "not_analyzed",
                      "doc_values": "true"
              },
              "begin": {
                "type": "integer", "index": "not_analyzed",
                    "doc_values": "true"
              },
              "end": {
                "type": "integer", "index": "not_analyzed",
                    "doc_values": "true"
```

```
90                },
91                "occurences": {
92                  "type": "object",
93                  "properties": {
94                    "group": {
95                      "type": "string", "index": "not_analyzed",
                            "doc_values": "true"
96                    },
97                    "begin": {
98                      "type": "integer", "index": "not_analyzed",
                            "doc_values": "true"
99                    },
100                   "end": {
101                     "type": "integer", "index": "not_analyzed",
                            "doc_values": "true"
102                   },
103                   "cui": {
104                     "type": "string", "index": "not_analyzed",
                            "doc_values": "true"
105                   }
106                 }
107               }
108             }
109           }
110         }
111       }
112     }
113   }
114 }
```

Listing 7: Index Schema

The table below describes the configuration for the article properties, as described in subsection 3.2.1. Each column represents:

Field
: The key in the key-value structure, it will contain a value with the properties defined in the indexing schema.

type
: The data type, in this case we only store Strings and Integers.

index
: It can possess three values:

  analyzed
  : The string is analyzed and only then it is indexed, this results in indexing the field as full text.

not_analyzed

The field is indexed, making it searchable, however it is indexed exactly as it is declared-

no

The field is not indexed, therefore it is not searchable.

doc_values

Doc values are a on-disk data structure that allows for faster sorting, aggregations and access to field values resorting to a data access pattern that looks up documents and finds the terms on a field, this structure is built at document index time. Although it makes entries occupy more disk space and is unusable on analyzed string fields, the performance enhancement it brings makes it such a staple in the ElasticSearch structure that there are future plans to make it a default feature in ElasticSearch, even for schemaless indexing.

Translation

In subsection 3.2.1 we listed the components that define a PubMed Article. In this column we make the translation between our stored field and the article's field.

Table 3.2: Data Mapping Definition

Field	type	analyzed	doc_values	Translation
Mesh	String	no	true	Mesh
Abstract	String	yes	false	Abstract
Authors	String	no	true	Authors
DOI	String	not indexed	false	Article Identifier
Jrn_title	String	no	true	Journal Title
Jrn_title_abbr	String	no	true	Journal Tile Abbreviation
Lang	String	no	true	Language
PMCID	String	no	true	Article Identifier
PMID	String	no	true	PMID
Title	String	yes	false	Title
Publ_year	Integer	no	true	Publication Year
Annotation	Object			Annotation List

Before explaining the nested layout of the annotations provided by Neji, it is important to understand the different methods through which ElasticSearch can store these types of structures. There are three distinct types one can map to store and index hierarchical data in ElasticSearch:

Nested Object

JSON documents are hierarchical by nature, that is, they may contain inner objects, however since Lucene has no concept of inner objects, ElasticSearch flattens object hierachies into a list of fields name and values, as demonstrated in listing 8. This results in losing associations between fields in the same inner object. In Listing 8 the association between *alice* and *white* is lost.

Nested as a datatype

In order to maintain the independence, the nested datatype is available. An object with the

nested type will have each object in the array indexed as a separate document, allowing for independent queries to each nested object, through the means of nested queries. This causes an increase in the number of stored documents and becomes very large when there are several levels of hierarchy.

Parent-Child

Parent-child relationships allows for document association, linking one document type with another, in a one-to-many association. It has some advantages over nested objects since:

- There is no need to reindex all children when the parent document is updated;
- Children can be added, removed or updated without affecting the parent or other children;
- Each child element can be returned as a result of a search query.

It brings a disadvantage since the parent document and its respective children must be indexed in the same shard. The mappings are stored in doc_values for better performance.

```
// Indexing
{
  "user": [
    {
      "first" : "John",
      "last" : "Smith"
    },
    {
      "first" : "Alice",
      "last" : "White"
    }
  ]
}
// Internal Representation
{
  "user.first" : ["alice", "john"],
  "user.last" : ["smith", "white"]
}
```

Listing 8: Object Flattening in ES (as in ElasticSearch documentation)

It was necessary to determine which datatype mapping would best fit Neji's hierarchical annotation format. The first method implemented considered each sentence in an annotation and each occurrence in a sentence as Nested, however this brought an unpleasant drawback, since the number of total documents in the index grew at a rate that was impossible to support, as evidenced by Table 3.3.

For a set of 121362 articles, if the indexing datatype was nested it would result in a total of almost 500 thousand documents, representing a value increase of approximately 41.09 times. Parent-Child relationships also index the parent and each individual child in a different document, this would mean that the discrepancy in the document count would be the same as with a nested datatype, therefore the Nested Object datatype was chosen.

health	status	index	pri	rep	docs.count	store.size
green	open	object	10	1	121362	1.2gb
green	open	nested	10	1	4986260	1.8gb

Table 3.3: Index information: Object vs Nested

Since, as shown in the previous section, the Neji annotation output can be represented in the JSON format the indexing on ElasticSearch is straight-forward. However, as said before, schema definition is needed for optimal performance. Table 3.4 illustrates the mapping definition for the Sentence Object Array stored in the fields **abstract_ann** and **title_ann**.

Field	type	index	doc_values	Translation
id	Integer	not_analyzed	true	Sentence id
Begin	Integer	not_analyzed	true	Sentence Begin
End	Integer	not_analyzed	true	Sentence End
Occurrences	Object			Annotation Array

Table 3.4: Annotations Mapping Definition

Each annotation occurrence can have multiple CUIs belonging to the same semantic group, there can be overlapping occurrences as long as they do not share the same semantic group. Table 3.5 illustrates the mapping of an annotation occurrence.

Field	type	index	doc_values	Translation
Group	String	not_analyzed	true	Semantic Group
Begin	Integer	not_analyzed	true	Occurrence Begin
End	Integer	not_analyzed	true	Occurrence End
Cui	String	not_analyzed	true	CUI Array

Table 3.5: Annotation Occurrence Mapping Definition

Redis is an open-source, in-memory data structure server [81]. It is another example of a NoSQL key-value data store, however, unlike a standard database, it is not disk-based. All data is kept on RAM so it is often used as a cache and a message broker.

Redis also has an advantage towards its competitors which is built-in persistence, meaning that it has the ability to save data to disk. This means that Redis can not only be used as a on-disk database but also that it will not lose data whenever it is restarted. Redis' datatypes can be simple key-value associations but also complex types such as hashes and lists.

In the scope of this project, Redis was not necessarily an essential component, however, we use it for three different purposes:

Concept Document Frequency
: Although this data can be queried through ElasticSearch, our co-occurence algorithm would not have an acceptable performance if we had to query ElasticSearch for each concept's document frequency, therefore we store it in Redis as a Integer and we fetch the document frequency for a batch of concepts.

CUI to term translation
: For performance purposes we keep the preferred term translation for each CUI in memory.

CUI to group translation
: Arguably, this could be stored in an ElasticSearch index. However, just like in the previous translation, we decided to only use ElasticSearch for expensive queries, allowing more important data to be loaded into Field Data.

We use an out-of-the-box configuration for Redis, since the data needs do not really ask for a complex tuning of the server. In Listing 9 the most important information provided by the command INFO, ran in a Redis Client connected to the server, is presented.

```
# Server
redis_version:2.8.4
tcp_port:6379

# Memory
used_memory_human:128.85M
used_memory_peak_human:159.90M
```

Listing 9: Redis Information

3.3 BACK-END

3.3.1 COMMUNICATION

The server-side of the application is responsible for establishing the connection between the databases and the client visualization platform. The server was built on top of Node.js [82], which is a run-time environment for asynchronous, event-driven, server side applications. As a server, it fits very well I/O intensive applications, such as this one. However, since it only works in a single-threaded environment it doesn't scale for a large number of users, neither is it designed for heavy computing.

The server implements two distinct communication methods, both using the HTTP protocol:

WebSockets

The WebSocket protocol was standardized in RFC 6455 [83], it implements a full-duplex connection over a TCP [84] port. It establishes a direct connection between the client's browser and the server. It is used in the context of this application to take advantage of Node.js' asynchronous, event-driven nature, allowing the client to send a request without having to forcefully wait for a response. After a request is sent by the client, Node.js takes care of all the interactions with the databases and when the result is ready the server pushes a message containing the result to the client. Not only does this method scale better than a RESTful API, but this type of architecture also allows, in the future, to create a collaborative environment between users. Since the scope of this project is directly related to the visualization component, most interactions with the server are implemented through this method.

RESTful API

RESTful services are the standard for communication nowadays, therefore, for the sake of publicly exposing some methods, a very simplistic RESTful API was implemented.

The Node.js server uses Express as a web framework to define specific routes and handle http requests. The server routing configuration is described in figure 3.3.

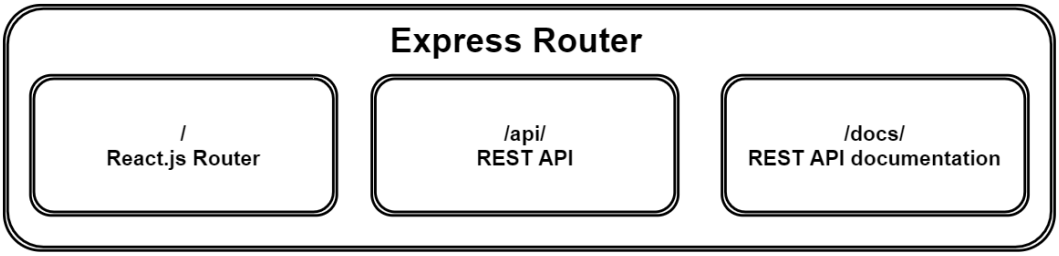

Figure 3.3: Server Routing

3.3.2 METHODS

Our system uses Socket.IO, a Javascript Library, to implement event-driven WebSocket communication. This library is implemented both in the client and server side. The most important methods defined (both in the server and the client) are illustrated on Table 3.6.

Id	Client Request	Server Response	Parameters	Task	Results
1	*suggest*	*suggestResponse*	Search query	Navbar Suggestions, through an ElasticSearch Suggest query	A list of string:CUI associations
2	*top_hits*	*hits_response*	Two CUIs representing biomedical concepts	Given two CUIs, search ES for documents where both CUI's must be found in the abstract and at least one of the concepts should be found in the title	List of matching articles, sorted by relevance
3	*cui2str*	*strResults*	A CUI representing a biomedical concept	Query ElasticSearch for the preferred term associated with a CUI	An object containing the association CUI->Term
4	*graph*	*graph_response*	A CUI representing a biomedical concept	Recursive algorithm that finds associated concepts based on Pointwise Mutual Information	JSON object containing a list of nodes and links

Table 3.6: WebSocket Methods

An in-depth explanation of each individual algorithm is described below, the code language is JavaScript and algorithm logic will be explained beforehand. In all functions there is a parameter named **`callback`**, this is the callback function to which the results will be passed.

1) Suggest

According to a search query, return 10 terms suggested by the ElasticSearch query.

```
1 // Sort ten shortest suggestions
2 function sortSuggestions(objs){
3     return obj.sort(smaller_than).slice(0, 10);
4 }
5
6 // Get suggestions
```

```
function suggestions(query, callback){
    // ElasticSearch Suggest Query
    var response = client.suggest({
       index: 'str2cui_obj',
       body:{
        convert_suggest:{
            text: query,
            completion:{
                field: "suggest",
                size : 200
            }
        }
       }
    });
    // Check for error
    if(response == error){
        callback(null)
    }
    else{
        callback(sortSuggestions(response.options));
    }
}
```

2) Top Hits

Given two distinct CUI, get all documents that obey the following rules:

- Both CUIs **MUST** be present in the abstract of the article;
- Both CUIs **SHOULD** be present in the title of the article;
- A **minimum** of one CUI **should** match in the title query.

```
// Top Articles for two distinct cui
function top_hits(cui_a, cui_b, callback){
  // ElasticSearch Bool Query
  var response = client.search({
    index: "pubmed_full",
    body:{
      query:{
        bool:{
          must:[
            {match: {"cui_lst": cui_a}},
            {match: {"cui_lst": cui_b}}
          ],
          should:[
            {match: {"title_ann.sentence.occurences.cui" :
               cui_a}},
```

```
15              {match: {"title_ann.sentence.occurences.cui" :
                    cui_b}}
16            ],
17            minimum_should_match: 1
18          }
19        },
20        size : 50
21      }
22    })
23    if(response == error){
24          callback(null)
25      }
26      else{
27          callback(response.hits);
28      }
29  }
```

3) CUI to String

Given a list of Objects (**selected**), each containing a CUI in the field **key**, query ElasticSearch's index **str2cui** on the field CUI to get matching terms. Afterwards, reduce the list to find the shortest matching term.

```
1  function cui2str(selected, callback){
2      // Get all cuis from list
3      var cui = selected.map((result, index) =>{
4         return result.key;
5      });
6      var response = client.search({
7          index: "str2cui_obj",
8          body:{
9              fields: ["str", "cui"],
10             size: 20,
11             query:{
12                 bool:{
13                     must:[{
14                         terms:{
15                             cui : cui
16                         }
17                     }]
18                 }
19             }
20         }
21     })
22     if(response == error){
```

```
        callback(null);
    }
    else{
        callback(
            response.hits.map(hit=>{
                return hit.str.reduce(shortest_term);
            });
        );
    }
};
```

4) Graph

Given a single CUI, call a recursive algorithm with a pre-defined depth of 2. In each iteration, ElasticSearch is queried to find the 250 concepts that most frequently co-occur with the node being evaluated at that time. The parameters are those nodes total document frequency and the document frequency of the co-occurence and they are used to calculate, through Pointwise Mutual Information, the 25 nodes that are more associated with the source node. Each new node is added to the list of nodes that will be evaluated. If a Node has a depth higher than the limit it is not added to the list.

```
function graph_correlation(cui, cb){

    var stats = client.indices.stats({index: 'pubmed_full'});
  var t = stats.indices.pubmed_full.primaries.docs.count;
  var ids = {};
  var nodes = [{"name": cui, "depth":0}]
  var links = [];
  ids[cui] = 0;
  recursive_correlation([[cui,0]], ids, nodes, links, 2, [], t,
      function(ended){
    get_node_names(nodes, function(ended){
      get_ranking(links, nodes, function(min, max, min_p, max_p){
          nodes[0].centrality = max;
        callback(
            "links" : links,
            "nodes" : nodes,
            "min_pr": min,
            "max_pr": max,
            "max_p":max_p,
            "min_p":min_p
        });
      });
    });
  });
```

```
24 };
25
26 function recursive_correlation(cui, ids, nodes, links, depth,
       closed, total, cb){
27   // If no more CUIs to evaluate return
28   if(cui.length == 0){
29     cb(True);
30   }
31   else if(cui[0][1] < depth){
32       // Current Cui and Current Depth
33     var cui_c = cui[0][0];
34     var current_d = cui[0][1];
35     // Add CUI to Closed Nodes
36     closed.push(cui_c);
37     // Get top 250 concepts associated to evaluated node
38     var relations = get_correlation(cui_c);
39     var cls = []
40     /*
41        ...
42        Push to list cls all closed nodes and nodes currently to
             be evaluated
43        ...
44       */
45     // Get top 25 nodes according to links' PMI, and add to
        existing links
46     var add = create_links(relations, cui_c, ids, links, total,
        current_d);
47     var to_add = [];
48     /*
49        ...
50        Push to list to_add all nodes in list add that aren't in
             list cls
51        Add to current nodes list the to_add list
52        Add to current cui list all cuis in list to_add with
             respective depth
53        ...
54     */
55       cui.shift();
56       var flag = recursive_correlation(cui, ids, nodes, links,
           depth, closed, total)
57         callback(flag);
58   }
59   else{
60     cui.shift();
61     var flag = recursive_correlation(cui, ids, nodes, links, depth,
```

```
            closed, total);
62        callback(flag);
63      }
64  };
```

3.3.3 DEPLOYMENT

In order to facilitate the development process and to optimize performance, a couple of additional libraries were installed in the server, such as:

Browserify
A compiler that gathers all dependencies and javascript files into one unique bundle.js file, allows for Node.js style require() in front-end pages.

Less
Less is a CSS pre-compiler, allowing for more complex rules in CSS such as class mixins (hierarchy) in CSS classes as well as defining nested rules and variables.

Babel
Transpiles ECMAScript 6 code [85] into runnable ECMAScript 5 javascript Code. ECMAScript 6 allows class definition in Javascript as well as functional programming, simplifying code organization and speeding up the development process.

Gulp
Gulp is an automation toolkit for running tasks defined in Javascript. It can automate tests, compile Less/Sass files, automatically re-bundle JS files when source is updated. In the context of this dissertation it has the following objectives:

- Compile Bootstrap, dependencies (through Browserify) and all JavaScript files into one unique bundle.js stored in the *public* folder;
- Update css and Javascript bundles when local files are changed;
- Automate Less Compilation, Babel Transpiler and Browserify.

3.4 FRONT-END

3.4.1 FRAMEWORKS

The front-end environment was designed to fit a modern web application design, therefore it includes state-of-the-art frameworks that enable it to deliver a complex visualization interface taking into account client variations such as screen size, hardware and browser.

Some key frameworks are used in the development of the front-end web application, such as `React.js` and `d3.js`. In the following section we will specify why both these frameworks were chosen over their competition and how they fit in the workflow of the web application GRACE.

REACT.JS

`React.js` is a Javascript Framework, created by Facebook, that provides a view layer for data, rendering it as HTML. It enables developers to built large applications with data that changes, since it only renders as a reaction to a change in data, allowing React.js components to only re-render data that has been modified. This allows the application to not be forced to re-render the whole view, as most User Interface frameworks do.

`React.js` has the advantage that it can be rendered in the server, which is faster than client-side rendering. It also provides a hierarchical association between components, allowing the existence of explicit parent-child relationships, meaning all nested components are children of the components that contain them. Components are the view unit of the system and have the advantage of being reusable.

The list of components and their connections is described in Figure 3.4, where arrows represent parent-child relations. `Props` are properties (variables) that are passed down from parent components to child components.

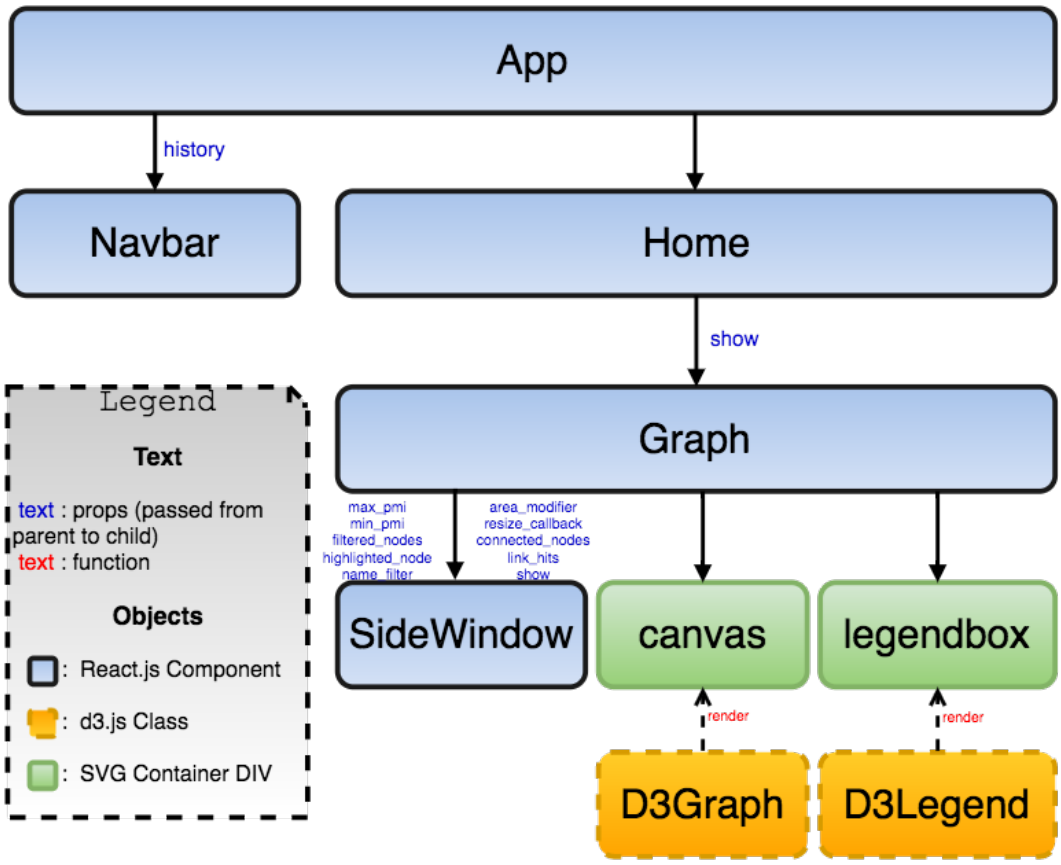

Figure 3.4: Front-end Component Hierarchy

Another important aspect of `React.js` is its data flow since we use Flux as the architecture. Flux is a pattern designed to utilize a unidirectional data flow, resorting to its main constituents:

Dispatcher
: The focal point that manages all data flow, since it distributes the actions to the stores. As a action is called, the dispatcher sends to the stores via callbacks defined when a store is registered.

Actions
: The dispatcher allows the application to trigger a dispatch which includes a payload of data, this is named an Action. An action may only be invoked from a controller view.

Stores
: These element contain a components' state and logic, they register themselves with the dispatcher and provide it with a callback. This allows an action to reach a store and its state to be updated. After this update, an event is broadcasted with the information that there has been a state change, so views that are dependent on that data may query the new state and update themselves.

Controller Views
: Views are responsible for rendering Html, these component receive data from the stores and are able to pass data down the chain of its descendants. Each view, when receiving an event from the store requests the updated data using the stores' get methods, afterwards it calls its own `setState`() method, causing the `render`() method to run.

The reasons why `React.js` stands out from its competitors is that as the application scales, the ability to only re-render views whose state has been modified is a great advantage. Also, the unidirectional data flow paradigm makes the development process easier to debug and assures, to a developer, complete control of the applications' life-cycle.

D3.JS

`D3.js` is a Javascript Library that allows developers to present documents in a myriad of different formats based on data. It has the advantage of not being bound to any proprietary framework, which means it can be used in conjunction with any other set of libraries and/or frameworks. `D3.js` uses HTML, SVG and CSS to present data, it also has a collection of transitions and transformations that allow visualizations to be interactive.

This framework runs purely on the client-side, leaving the server only responsible for serving the data used. We use D3's Force API to create force-directed graph layouts. It features elements such as a repulsive charge between nodes, a pseudo-gravity that brings nodes to the center of the visible area and geometric constraints for links. There is also the possibility of creating custom constraints and forces through the "tick" event, an event that is trigger whenever there is a modification on the layout.

Our layout algorithm was heavily modified throughout the implementation process, initially the graph was inspired by online examples, however, we found that most graph implementation examples had one of two problems: Too many or too little features. It was necessary to filter the features made available by `D3.js`, defining which would be beneficial for user interaction and which, even if they were great visual assets, would end up being an hindrance to the user's experience.

Our final layout implements the following features:

Force

Our force layout has a pre-defined charge equal for all nodes, yet the links' distance and strength are inherently dependent on each nodes' depth, creating a layout that is capable of aggregating nodes by depth.

```
1
2  var force = d3.layout.force()
3                       .charge(-100)
4                       .linkDistance(distance)
5                       .linkStrength(function(d){
6                          if(d.depth === 4)
7                              return 0.6;
8                          if(d.depth === 2)
9                              return 0.8;
10                         return 1
11                      })
12                      .size([width, height])
13                      .alpha(0.2);
14
15 function distance(d){
16   if(d.depth ===2 ){
17     return d.depth * d.depth * 40;
18   }
19   else if(d.depth === 1){
20     return 50;
21   }
22   else if(d.depth ===4){
23     return height/4;
24   }
25 }
```

Color Coded

Nodes are color coded according to the semantic group to which they belong to.

Pan and Zoom

In order to ease user interaction, pan and zoom features were added.

Loading in background

The force layout causes objects to move until they stabilize, we noticed that this caused less performant computers to freeze when trying to render the moving force layout, therefore we let the graph stabilize in the background and only display it after we stop the force.

Mouse Events

Mouse events such as `mouse hover` and `mouse clicks` were added to allow users to change the

view, giving a greater detail for each individual node and links.

Node Highlighting

Whether it is individual highlighting or group highlighting, this feature allows the user to identify single nodes when searching for them and identify node associations.

`D3.js`' biggest strength is also its major flaw, since it offers a world of possibilities for developers to display data. The overwhelming number of features may be tempting to experiment with, but when it comes down to defining a user interaction approach one must identify what is absolutely necessary and leave out futile embellishments.

CHAPTER 4

Results

This section tries to demonstrate this work's value by displaying the project's results, providing an analysis of the Graphical Interface implemented as well as a performance analysis of the system.

The goal of this dissertation was to build a system capable of identifying and storing biomedical concept associations found on MEDLINE publications, offering a graphical interface for data exploration. Therefore, the result of this dissertation is presented as a web application named GRACE (GRAphical Concept Explorer). The application provides a graphical interface in which users can explore word association graphs constructed based on extracted information from MEDLINE articles.

4.1 GRACE

The GRACE application offers the following features:

- Explore biomedical concept association through a interactive graph display;
- Intuitive data visualization allows users to quickly determine relevant concepts based on a queried term;
- View each link's association strength;
- Get each node's centrality based on a page rank algorithm;
- Retrieve a list of MEDLINE publications that best document each concept-pair association;
- Read, filter and explore MEDLINE publications.

4.1.1 HOME PAGE

GRACE's home page provides an overview of the features made available in the web application and presents a navigation bar that contains the search input. This input's placement and functionality

is constant throughout the whole application.

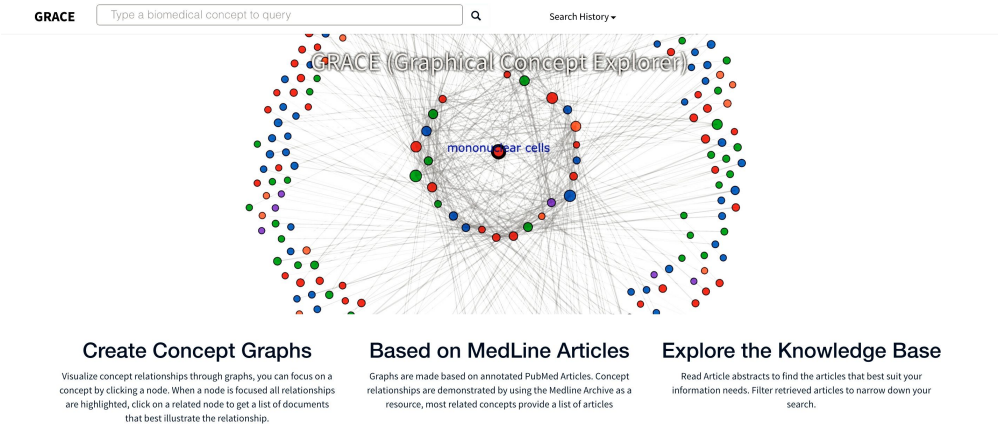

Figure 4.1: GRACE's Home Page

The search input has an auto-complete feature, suggesting terms that are stored in the ElasticSearch index, using the "suggest" method described in Table 3.6, suggestion results are shown in Figure 4.2.

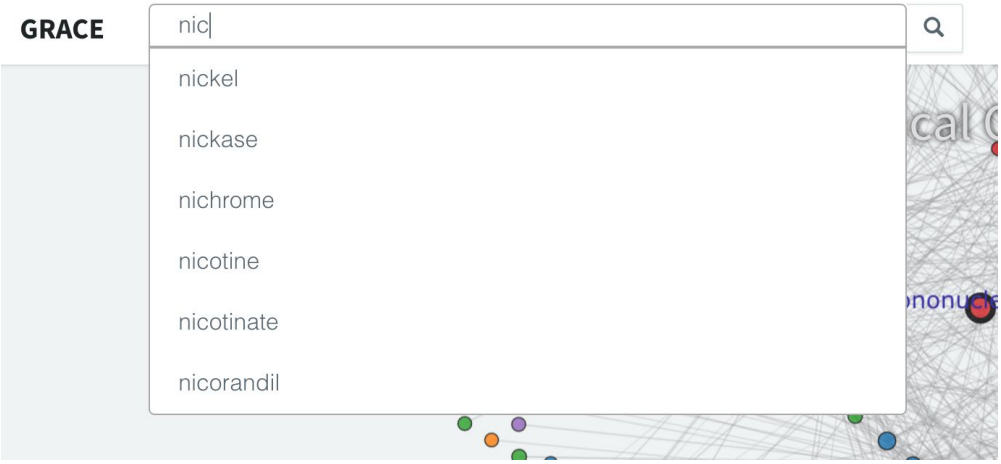

Figure 4.2: GRACE's Search Suggestions

4.1.2 GRAPH VIEW

When a biomedical concept is queried through the search input, a graph centered on it is presented to the user after a brief loading. The graph display arranges nodes in three different zones: **a)** The root node is in the center of the graph; **b)** Nodes directly connected to the root are positioned in an inner ring around the center node; **c)** Other nodes are organized around in an outer ring. This display is illustrated in Figure 4.3.

This view can be considered the default state of the graph, a number of actions can be taken at this point, as evidenced in Figure 4.4.

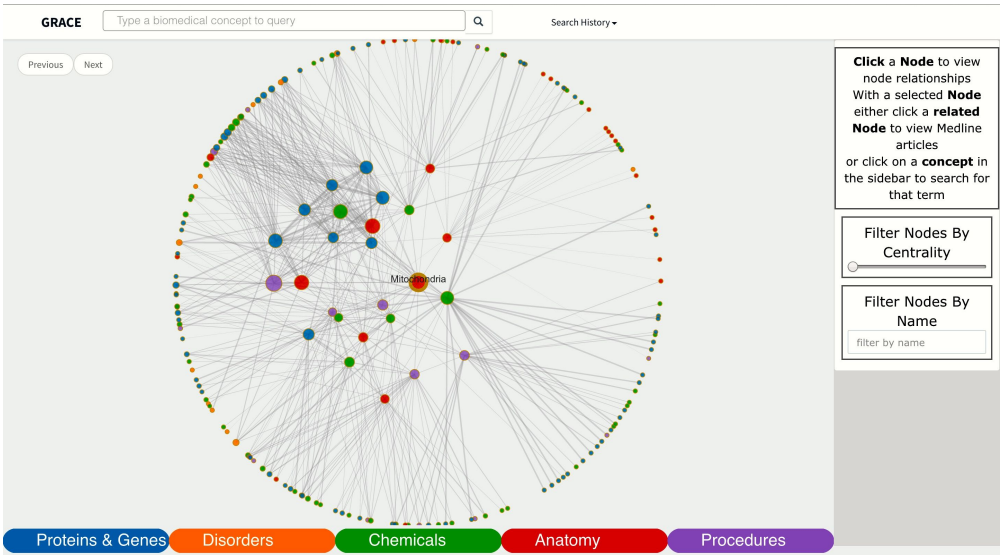

Figure 4.3: GRACE's Graph - Default State

Graph Interaction

The graph display is the central figure of the view, due to its relevance it is obviously interactive. A user is able to access other views by hovering or clicking on a node.

Filtering

Using the controls on the right side of the screen, the user is capable of filtering nodes by their centrality measure (given by a Page Rank algorithm) or by their name.

Navigation through history

If there is a search history of graphs a user is capable of moving from graph to graph by accessing the search history drop-down menu or using the `Previous` and `Next` arrows.

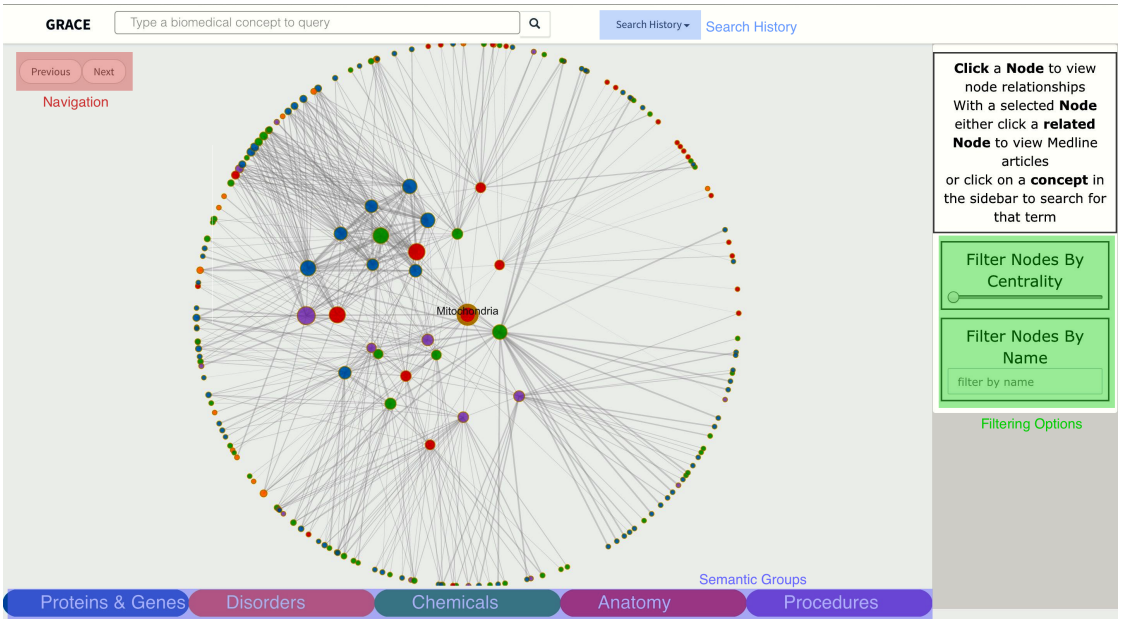

Figure 4.4: GRACE's Graph

The `Centrality Filter`, illustrated in figure 4.4 on the `Centrality Option` sections, allows users to differentiate nodes based on their centrality ranking, using a slider. This process hides nodes with a centrality ranking below the percentage defined by the slider. When the centrality slider is on the left-most side, the percentage is 0%, therefore all nodes are displayed. Contrastingly, when the slider is on the right-most side, the percentage is 100% making it so that only nodes with maximum ranking visible. These events are illustrated in detail in Figure 4.5.

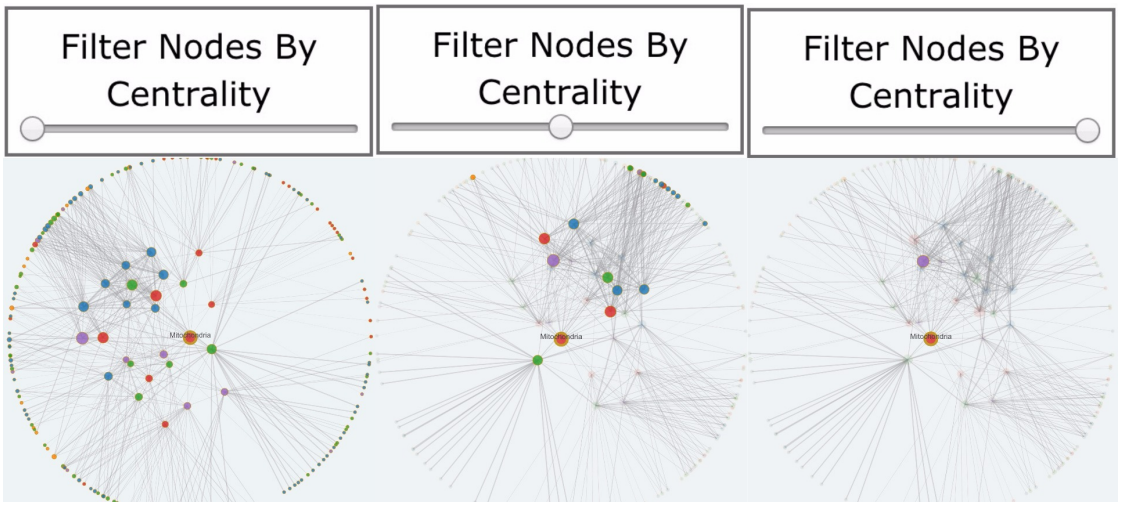

Figure 4.5: GRACE's Centrality Filter

Likewise, the `Name Filter`, located below the centrality slider, allows users to search the graph results for a specific concept, with auto-complete suggestions based on the concept names that exist on the graph. It will hide all unmatched nodes for a short amount of time.

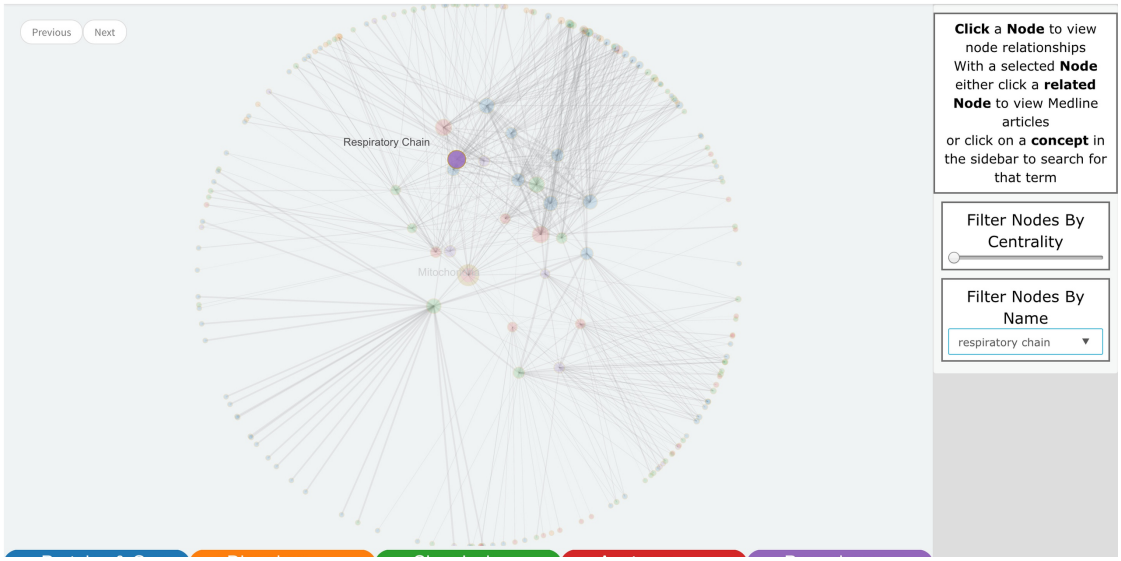

Figure 4.6: GRACE's Name Filter

The `Search History` drop-down menu allows users to access graphs from previously queried concepts as shown in Figure 4.7, this is also possible using the `Previous` and `Next` buttons on the top-left side of the screen.

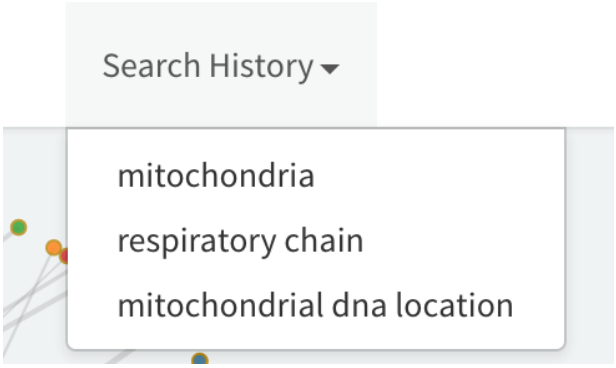

Figure 4.7: GRACE's Search History

4.2 GRAPH INTERACTION

4.2.1 DEFAULT STATE

Assuming the graph's default state, there are five types of interaction that a user can have with the display:

Hover over a node

Hovering over a node changes the focus of the graph to the hovered node, this causes the selected node and all nodes associated with it to become highlighted as seen in Figure 4.8. The sidebar is also affected, displaying the selected node's name, its centrality and the list of associated nodes sorted by the association measure. Each associated node's information contains the node's name and a bar with length proportional to the association's strength and color corresponding to the node's semantic group. This detailed view is removed as the user leaves the node's area with the mouse.

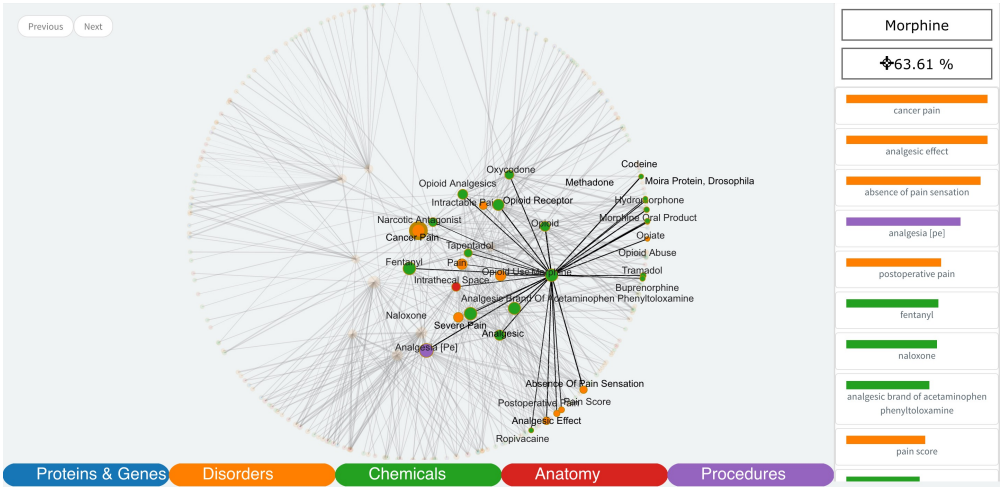

Figure 4.8: GRACE's Graph Node Hover View

Left-Click a node

The result of left-clicking a node is similar to hovering, however, the node selection is locked. Left-clicking a node changes the graph's state, transitioning into a view focused in the selected node and enabling a new set of actions. This state can be observed in Figure 4.9.

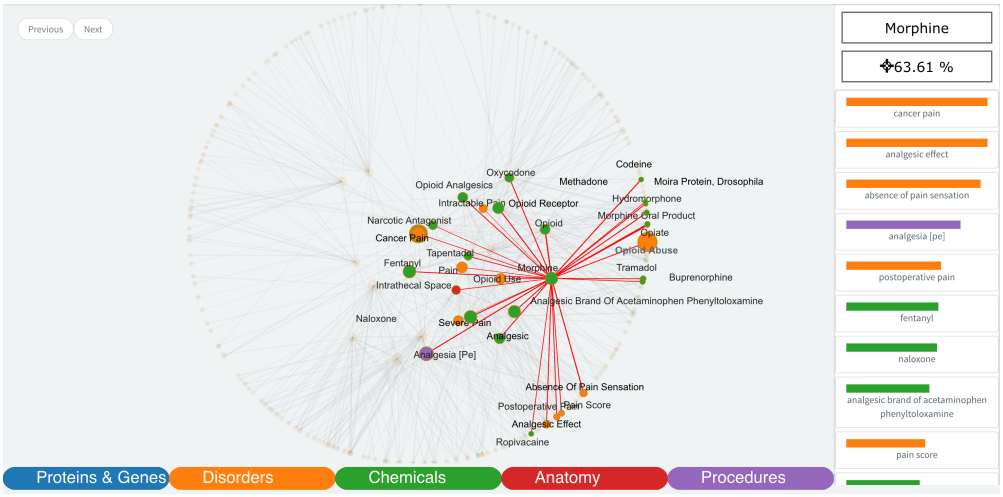

Figure 4.9: GRACE's Graph Node Selected State

Right-Click a node

Right-clicking a node displays a context menu, as seen on Figure 4.10, that allows the user to query the system for that concept, centering the graph on the select node.

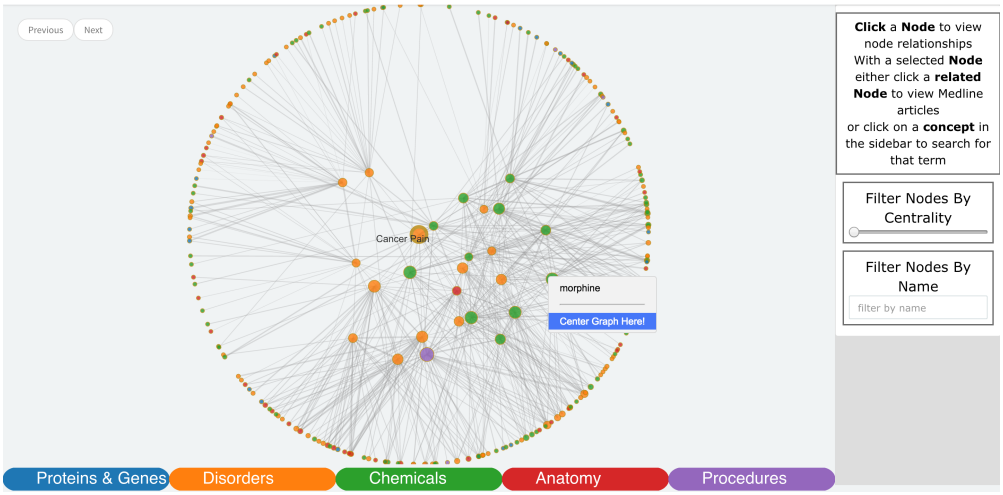

Figure 4.10: GRACE's Graph Node Right-Click

Drag

The graph display is *draggable*, the user can do this action by holding the left-button of the mouse in any area of the display. Moving the mouse around while holding the left-button will drag the graph around, allowing users to position the graph to their liking.

Zoom

There is also a zooming function that allows, through scrolling, the user to zoom the graph display in and out, according to the scroll movement.

4.2.2 NODE SELECTED STATE

As a node is selected and the focused view is locked, a user can interact with the application using the following set of actions:

Hover an associated node

Hovering an associated node's text or circle will highlight it, increasing its size relatively to other nodes. This function is enabled in order to facilitate the selection of smaller nodes. An example is seen of Figure 4.11.

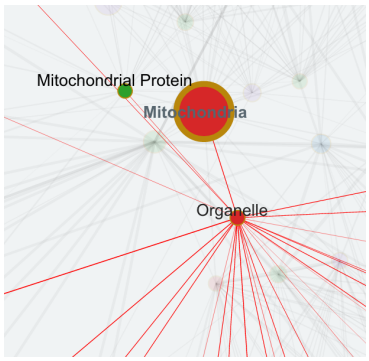

Figure 4.11: GRACE's Associated Node Highlighting

Click an associated node

Clicking on an associated node transitions the application into a third state where the display is focused in a specific association. This new view only displays the nodes that are involved in the association. It also updates the side bar, displaying a list of articles that best describe the association, using the "top_hits" method depicted in Table 3.6. The new view is represented in Figure 4.12.

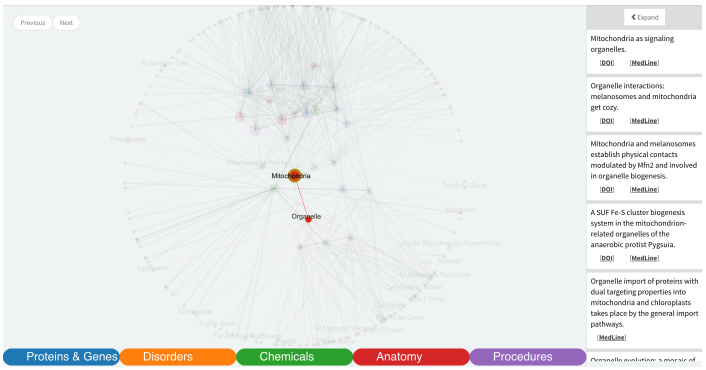

Figure 4.12: GRACE's Association View

Hover over a node in Sidebar

Hovering over a node in the sidebar highlights the node in the same manner as if hovered on the graph display

Click a node in Sidebar

Clicking a node in the sidebar queries the system using that node's identifier and presents a new graph centered on the clicked node.

Click the selected node

Clicking the selected node makes the visualization transition back into the graph's default state.

4.2.3 ASSOCIATION SELECTED STATE

This view is focused in presenting relevant information based on the selected association, however, before explaining the options offered by the article browser it is important to note that the user can still interact with the graph display.

Click the origin node

Clicking the node that was selected in the first will revert the graph to its default state.

Click the associated node

Clicking the associated will revert the graph to the node selected state.

4.3 MEDLINE ARTICLE EXPLORER

GRACE's article explorer has two distinct views, it can either be condensed, allowing the user to view the graph display and to have an overview of the articles that describe the selected nodes' relationship or extended, making the article explorer occupy most of the screen and allowing for extensive research on the articles' content.

The article explorer appears, first off, when the graph display is in the Association Selected State. The user is unable to access the article viewer without the graphical display being on said state.

4.3.1 CONDENSED ARTICLE EXPLORER

When the explorer is on this state, as seen on Figure 4.12, it presents very little information about each article, only providing the article's title and links, if they exist. This overview gives the user the possibility of continuing the graphical interaction, while accessing interesting articles, without the need to suppress the graph area.

However, this may not be enough to determine if an article's information is relevant to the user's information needs, therefore an extended view for the article explorer was created.

4.3.2 EXTENDED ARTICLE EXPLORER

The extended article explorer displays a list of articles containing all relevant information regarding each article. The list of article attributes that can be retrieved using the application are, as illustrated in Figure 4.13.

- Title
- Abstract sentences that contain the selected associations

- Authors
- Publication Year
- Links
- Full Abstract

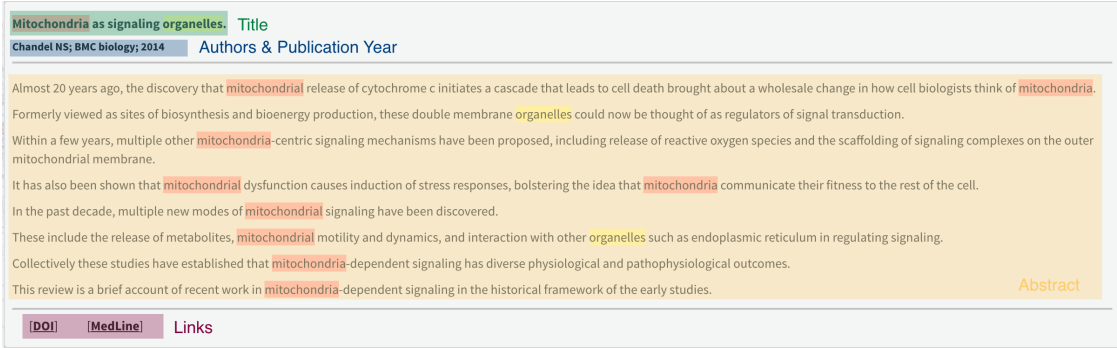

Figure 4.13: Article

The extended article explorer's view is illustrated in Figure 4.14. Both of the associated nodes' annotations are highlighted in the title and abstract, however in the abstract only sentences that contain at least one of these annotations are presented.

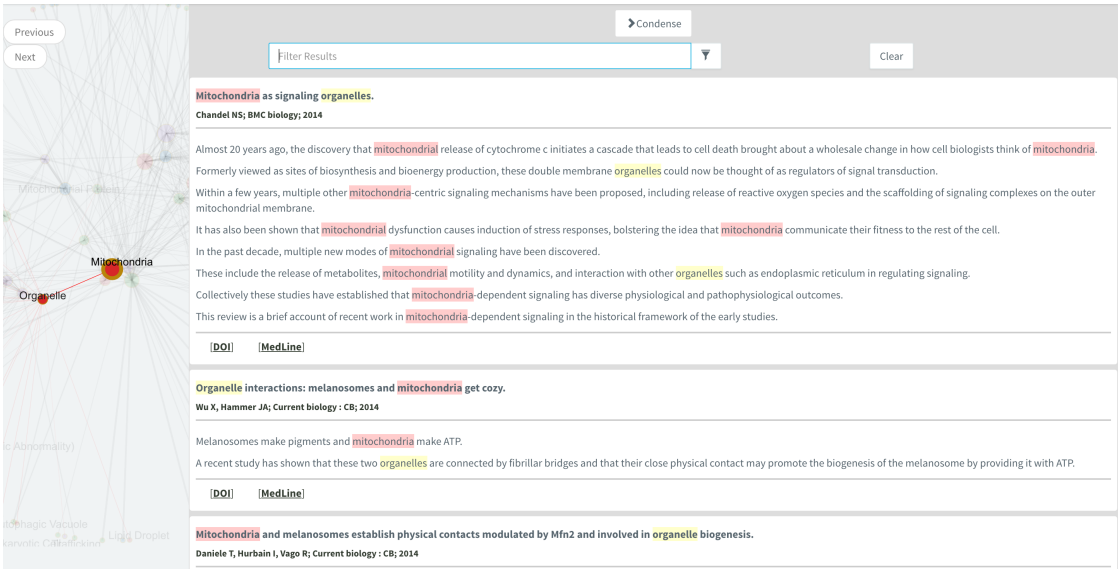

Figure 4.14: GRACE's Extended Article Explorer

In this view the system has implemented the following features:

Show Full Abstract

Clicking the authors and publication year causes the full abstract of the article to appear as a HTML Modal, as seen on Figure 4.15.

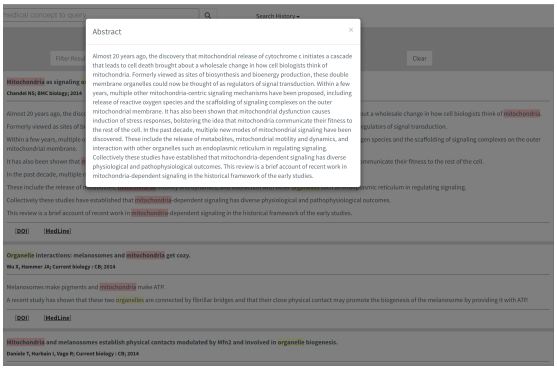

Figure 4.15: Article's Full Abstract Modal

Sentence Filtering

Using the filter input, the system utilizes the query input defined by the user to perform full-text search on the sentences displayed below in order to allow a simple method for users to scavenge through the contents of different articles for specific information.

Go to article reference

Clicking the HyperLinks in the bottom-left of any article will open a new browser tab with the original abstract in Pubmed or full-text of the article, if available.

Condense Article Explorer

Clicking the condense button on the top of the Article Explorer Window will return the user to the graph interaction, maintaining the Association Selected State.

4.4 API

GRACE provides a RESTful API that can be used to query the system without using the web application. Although it is concise, it expose the most important methods implemented by the system. It is documented and its documentation is available on the web application. The REST API provides three main methods, described in Table 4.1.

Method	Parameters			Url
	Field	Type	Description	
Correlation	cui	String	The UMLS identifier of the concept	/api/correlation
Graph	cui	String	The UMLS identifier of the concept	/api/graph
	depth	Integer	The maximum depth for the algorithm	
Top Hits	cui_a	String	The UMLS identifier of the first concept	/api/top_hits
	cui_b	String	The UMLS identifier of the second concept	

Table 4.1: RESTful API Methods

An example for each method is described in Listing 10

```
1 Method = Correlation
```

```
url = /api/correlation?cui=<cui>

Response:
HTTP/1.1 200 OK
{
  "co_occurences":[
    {
      "key": "CUI_Identifier",
      "co_occurences": 75, // Mutual Occurence of both concepts
      "occurences_y": 5257, // Occurences of concept y
      "pmi": 0.120293545 // Calculated PMI
    }
  ],
  "occurences_x": 3548 //Occurences of queried concept

}

Method = Graph
url = /api/graph?cui=<cui>&depth=<max_depth>

Response:
HTTP/1.1 200 OK
{
  "links":[
    {
      "source": 0, // Index of source node
      "target": 1, // Index of target node
      "pmi": 0.7644234827134753, // PMI of connection
      "depth":1 // Depth of the source node of the link
    }
  ],
  "nodes":[
    {
      "name": "C1231230", // Node CUI Identifier
      "radius": 30,     // Default Radius for d3.js visualization
         (optional)
      "depth": 1,       // Node Depth
      "str": "some concept", // The Concept name normalization
      "group", "DISO",      // Concept's Semantic Group
      "centrality": 0.004123  // Node Centrality based on weight
         Page-Rank Algorithm
    },
    {
      "name": "C3234112",
      "radius": 30,
```

```
      "depth": 2,
      "str": "some other concept",
      "group": "CHEM",
      "centrality": 0.006423
    }
  ],
  "min_pr": 0.003,   // Minimum value for PageRank
  "max_pr": 0.007,  // Max value for PageRank
  "max_p": 1,       // Max PMI value
  "min_p": 0.2777   // Min PMI value

}

Method = Top Hits
url = /api/top_hits?cui_a=<first_cui>&cui_b=<second_cui>

Response:
HTTP/1.1 200 OK
{
  [List of articles sorted by Retrieval Score]
}
```

Listing 10: REST API examples

4.5 PERFORMANCE

In order to determine GRACE's back-end performance, some tests were implemented to prove GRACE's scalability and stability. These tests are obviously influenced by the characteristics of the system they are executed in. Therefore, the system specs of the test machine are represented below.

CPU Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz, 12 Cores;

RAM 192GB @ 2133 MHz;

Disk 4TB HDD.

Additionally, the environment of the tests plays a role on the results, for this reason, the deployment environment is described below.

- Service runs on a Docker container with elevated permissions;
- Number of maximum open file descriptors increased;
- Swap memory usage is disabled;
- Node.js server has a maximum stack size of 32 thousand;

- ElasticSearch Cluster running with two nodes, each of them with a heap size of 10Gb;
- ElasticSearch Index has 10 primary shards and 1 replica shard for each primary;
- At the moment of writing, 1,833,036 articles are indexed.

Since Elasticsearch caches queries results, it is necessary to discern results that utilize caching from those that do not. Therefore, all tests that are presented contain these two variants. The conducted tests utilize the REST API as a mean of communication, the retrieval time is given by the processing time of the server, resorting to an analysis of the Node.js server's logging file.

The method ***graph***, that produces a graph of relationships consisting of nodes and links, given a certain CUI and a depth value, which determines the maximum value of a node's depth, and the method ***top_hits***, that returns the articles that best describe a relationship between two different CUI's will have their performances evaluated. In order to test these methods, the 20 concepts that occur most frequently in the collections were used as parameters for the tests.

For the ***graph*** method, each one of this concepts will be queried with different depths (1, 2 and 3). The number of retrieved nodes, links and the response time will be taken into account. Results are presented in Figure 4.16 (without resorting to Elasticsearch's cache functionality) and Figure 4.17 (utilizing Elasticsearch's cache) and also in Table 4.2.

Figure 4.16: (Retrieval Time-Node) relation with empty cache

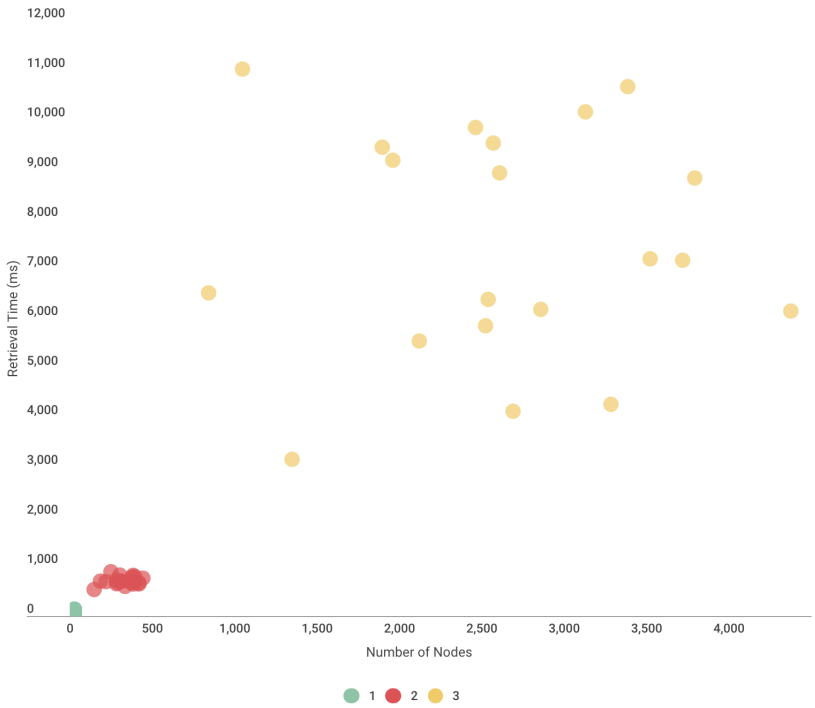

Figure 4.17: (Retrieval Time-Node) relation with cache

Depth	1		2		3	
Metrics	Mean	σ	Mean	σ	Mean	σ
Retrieval Time no cache (ms)	266.3	64.72	1041.0	102.95	8086.4	2387.95
Retrieval Time w/ cache (ms)	101.54	44.22	718.47	82.41	7516.3	2267.21
Number of Nodes	26	0	320	79.77	2632.75	903.48
Number of Links	25	0	598.95	35.97	6927.55	1905.73

Table 4.2: System performance statistic according to depth

Analyzing the data represented above we can conclude that using Elasticsearch's cache enhances the system's performance. Therefore, it is important that the Heap Size allocated to Elasticsearch's Java Virtual Machine container is large enough so that it can store a considerable amount of query results in order to speed up search times. It is also possible to determine that the number of nodes and links increases exponentially. The graphical interface only queries the system for graphs with depth 2, in order to maintain a balance between the quantity of information and the time needed to retrieve said information.

For the ***top_hits*** method, we used a k-combination of the collection of the ten most frequent terms, with k = 2, producing 45 concept pairs that were queried. The size of the response and its time will be taken into account. Results are presented in Figure 4.18.

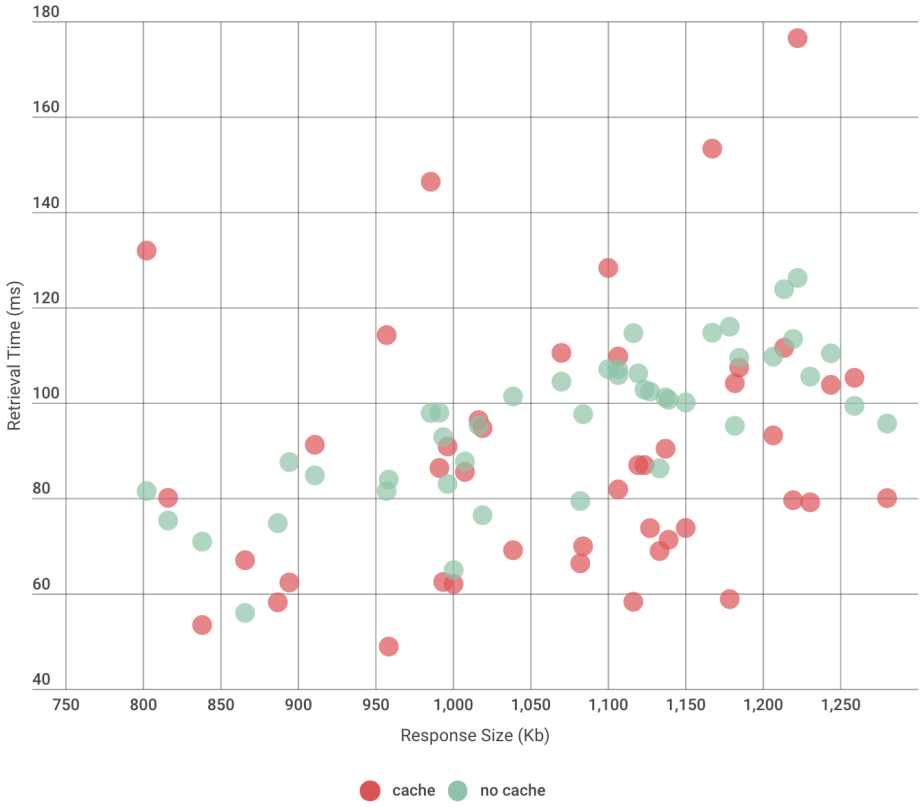

Figure 4.18: Top Hits method performance

Analyzing Figure 4.18 we can see that, for the most part, utilizing Elasticsearch's cache reduces the retrieval time, especially when retrieving larger documents.

CHAPTER 5

Conclusion

The main goal of the work was to study, design and develop a solution that is able to identify, through a trustworthy measure, associated concepts present in the MEDLINE articles and provide an user-friendly application that allows users to fulfill their information needs.

Initially an intensive study of available tools and approaches was made, as well as an in-depth technological investigation in order to find the components that would best fit the final solution. As a result we developed GRACE, a web-based platform for graphical exploration of biomedical concept associations.

The user interface provides a large number of features, allowing the users, through simple interactions with the graphical display, to navigate the biomedical domain in search for relevant information found in articles or through the discovery of latent associations in the graph. Additionally, it integrates an article explorer that can be used to find relevant references for biomedical academic work.

GRACE's applicability is given by the successful integration of a graphical solution with a robust and complex processing infrastructure. The objective of this system was to provide a technologically advanced and reliable structure that grants the possibility of being extended in the future.

Some improvements can be made in order to improve the system reliability such as improving the Word Sense Disambiguation step of the annotation process and reducing "noise" provided by unreliable dictionary entries. Additionally, several useful extensions can be integrated in the applications such as: **a)** Adding concept relation types, extending the annotation system to provide events between concept annotations; **b)** The possibility of dynamically creating and editing a graph to the user's preference; **c)** Providing a citation exporter/manager in the article explorer; **d)** Adding the possibility to create sub-collections from the base collection (all MEDLINE articles with an English abstract); **e)** Allowing different associations measures to be used in the graph construction.

We feel that GRACE provides top-notch performance and an intuitive graphical interface with several features that enhance the user's freedom, providing an advantage towards other related implementations. This advantage is brought by three main factors and although the objective of

this dissertation is not to prove that GRACE does necessarily surpass its competition in all of them, users may find one of its strong points to be either its scope dimension, identifying concepts from several different semantic groups, the fast delivery of results or the intuitive and modern graphical interface.

Overall, GRACE presents several advantages to the biomedical community, providing an useful tool for researchers looking for associations between concepts in the biomedical domain, whether they are explicit or latent. This is achieved through a knowledge visualisation and navigation paradigm that is highly responsive, simple to use, and engaging.

References

[1] J. Shotwell, *An Introduction to the History of History*, ser. Records of civilization. Columbia University Press, 1922. [Online]. Available: `https://books.google.pt/books?id=R7YNAAAAIAAJ`.

[2] R. Blumberg and S. Atre, "The problem with unstructured data", *DM Review*, vol. 13, pp. 42–49, 2003.

[3] M. A. Hearst, "Untangling text data mining", in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ser. ACL '99, Association for Computational Linguistics, 1999, pp. 3–10.

[4] M. Beaulieu, "Interaction in information searching and retrieval", *Journal of Documentation*, vol. 56, no. 4, p. 431, 2000.

[5] R. Gaizauskas and Y. Wilks, "Information extraction: Beyond document retrieval", *Journal of Documentation*, vol. 54, no. 1, pp. 70–105, 1998.

[6] L. Hirschman, "The evolution of evaluation: Lessons from the message understanding conferences", *Computer Speech & Language*, vol. 12, pp. 281–305, 1998.

[7] D. A. Lindberg, "Internet access to the National Library of Medicine", *Eff Clin Pract*, vol. 3, no. 5, pp. 256–260, 2000.

[8] D. Campos, S. Matos, and J. L. Oliveira, "Current methodologies for biomedical named entity recognition", in *Biological Knowledge Discovery Handbook: Preprocessing, Mining and Postprocessing of Biological Data*, M. Elloumi and A. Y. Zomaya, Eds., John Wiley & Sons Inc., Dec. 2013, pp. 839–868.

[9] A. R. Aronson, "Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program", *Proc AMIA Symp*, pp. 17–21, 2001.

[10] "Ncbo annotator: Semantic annotation of biomedical data", in *8th International Semantic Web Conference (ISWC2009)*, Oct. 2009.

[11] M. Tanenblatt, A. Coden, and I. Sominsky, "The conceptmapper approach to named entity recognition", in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010, pp. 19–21.

[12] D. Campos, S. Matos, and J. L. Oliveira, "A modular framework for biomedical concept recognition", *BMC Bioinformatics*, vol. 14, p. 281, Jan 2013.

[13] O. Bodenreider, "The Unified Medical Language System (UMLS): integrating biomedical terminology", *Nucleic Acids Res.*, vol. 32, no. Database issue, pp. D267–270, Jan. 2004.

[14] N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M. A. Storey, C. G. Chute, and M. A. Musen, "BioPortal: ontologies and integrated data

resources at the click of a mouse", *Nucleic Acids Res.*, vol. 37, no. Web Server issue, W170–173, Jul. 2009.

[15] K. Sagae and J. Tsujii, "Dependency parsing and domain adaptation with lr models and parser ensembles", in *Proceedings of the CoNLL 2007 Shared Task in the Joint Conferences on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic, 2007, pp. 1044–1050.

[16] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data", in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[17] A. K. McCallum, "Mallet: A machine learning for language toolkit", http://www.cs.umass.edu/ mccallum/mallet, 2002.

[18] A. S. Schwartz and M. A. Hearst, "A simple algorithm for identifying abbreviation definitions in biomedical text", in *Pacific Symposium on Biocomputing*, 2003, pp. 451–462.

[19] brat contributors. (2014). Standoff format - brat rapid annotation tool, [Online]. Available: `http://brat.nlplab.org/standoff.html`.

[20] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, ser. CONLL '03, Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 142–147. DOI: `10.3115/1119176.1119195`. [Online]. Available: `http://dx.doi.org/10.3115/1119176.1119195`.

[21] D. Crockford, "Rfc 4627 - the application/json media type for javascript object notation (json)", Tech. Rep. [Online]. Available: `http://tools.ietf.org/html/rfc4627`.

[22] Y. Tsuruoka, J. McNaught, and S. Ananiadou, "Normalizing biomedical terms by minimizing ambiguity and variability", *BMC Bioinformatics*, vol. 9, no. 3, pp. 1–10, 2008.

[23] E. Agirre and P. Edmonds, *Word Sense Disambiguation: Algorithms and Applications*, 1st. Springer Publishing Company, Incorporated, 2007, ISBN: 1402068700, 9781402068706.

[24] N. Ide and J. Véronis, "Introduction to the special issue on word sense disambiguation: The state of the art", *Comput. Linguist.*, vol. 24, no. 1, pp. 2–40, Mar. 1998.

[25] M. Weeber, J. G. Mork, and A. R. Aronson, "Developing a test collection for biomedical word sense disambiguation", *Proc AMIA Symp*, pp. 746–750, 2001.

[26] J. Pustejovsky, J. Castaño, R. Saurí A. Rumshinsky, J. Zhang, and W. Luo, "Medstract: creating large-scale information servers for biomedical libraries", in *Proceedings of the ACL-02 workshop on Natural language processing in the biomedical domain*, Morristown, NJ, USA: Association for Computational Linguistics, 2002, pp. 85–92.

[27] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Wokingham, UK: Addison-Wesley, 1999.

[28] E. M. Voorhess, "The efficiency of inverted index and cluster searches", in *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, USA, 1986, pp. 164–174.

[29] E. M. Voorhees, "Natural language processing and information retrieval", in *Information Extraction: Towards Scalable, Adaptable Systems*, 1999, pp. 32–48.

[30] T. S. Donald Metzler and W. B. Croft, *Search Engines: Information Retrieval in Practice*. Pearson PLC, 2009.

[31] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*. Prentice Hall PTR, Jun. 1992.

[32] D. Hiemstra, "Information retrieval models", in *Information Retrieval: Searching in the 21st Century*, A. Goker and J. Davies, Eds., Wiley, Oct. 2009. [Online]. Available: `http://eprints.eemcs.utwente.nl/13527/`.

[33] A. Singhal, "Modern information retrieval: a brief overview", *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, pp. 35–42, 2001.

[34] D. Lee and K. Seamons, "Document ranking and the vector-space model", *IEEE Software*, vol. 14, pp. 67–75, 1997, ISSN: 07407459. DOI: `10.1109/52.582976`.

[35] H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information", *IBM J. Res. Dev.*, pp. 309–317, Oct. 1957.

[36] S. ROBERTSON, "The probability ranking principle in ir", *Journal of Documentation*, vol. 33, no. 4, pp. 294–304, 1977.

[37] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms", *Journal of the American Society for Information Science*, no. 3, pp. 129–146, 1976.

[38] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3", 1996, pp. 109–126.

[39] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval", in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '98, Melbourne, Australia, 1998, pp. 275–281.

[40] S. Büttcher, C. Clarke, and G. V. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines*. The MIT Press, 2010.

[41] Apache, *Apache solr*, `http://lucene.apache.org/solr/`, (Accessed on 06/01/2016), 2016.

[42] Elastic.co, *Elasticsearch | elastic*, `https://www.elastic.co/products/elasticsearch`, (Accessed on 06/01/2016), 2016.

[43] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Greenwich, CT, USA: Manning Publications Co., 2010.

[44] C. Middleton and R. Baeza-yates. (2007). A comparison of open source search engines, [Online]. Available: `http://wrg.upf.edu/WRG/dctos/Middleton-Baeza.pdf`.

[45] LucidWorks. (Jan. 21, 2012). Who is using lucene/solr, [Online]. Available: `https://lucidworks.com/blog/2012/01/21/who-uses-lucenesolr/`.

[46] DB-Engines. (2016). Db-engines ranking of search engines, [Online]. Available: `http://db-engines.com/en/ranking/search+engine`.

[47] Elastic. (2015). Elasticsearch reference - getting started - basic concepts, [Online]. Available: `https://www.elastic.co/guide/en/elasticsearch/reference/2.0/_basic_concepts.html`.

[48] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*. O'Reilly Media, Inc., 2013.

[49] B. S. Everitt and A. Skrondal, *The Cambridge Dictionary of Statistics*, 4th. Cambridge, UK; New York: Cambridge University Press, 2010.

[50] G. Upton and I. Cook, *A Dictionary of Statistics*, 3rd. Oxford University Press, 2014.

[51] P. Pecina, "Lexical association measures and collocation extraction", *Language Resources and Evaluation*, vol. 44, no. 1, pp. 137–158, 2010.

[52] T. Dunning, "Accurate methods for the statistics of surprise and coincidence", *Comput. Linguist.*, vol. 19, no. 1, pp. 61–74, Mar. 1993.

[53] A. Edwards, *Likelihood*, ser. Cambridge science classics. Cambridge University Press, 1984.

[54] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography", *Comput. Linguist.*, vol. 16, no. 1, pp. 22–29, Mar. 1990.

[55] R. Fano, *Transmission of Information: A Statistical Theory of Communications*. Cambridge, MA: The MIT Press, 1961.

[56] K. Pearson, *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to Have Arisen from Random Sampling*. 1900.

[57] S. K. Card, J. D. Mackinlay, and B. Shneiderman, Eds., *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.

[58] D. A. Norman, *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. 1993.

[59] B. Tversky, J. B. Morrison, and M. Betrancourt, "Animation: Can it facilitate?", *International Journal of Human-Computer Studies*, vol. 57, pp. 247–262, 2002.

[60] E. R. Tufte, *The Visual Display of Quantitative Information*. Cheshire, CT, USA: Graphics Press, 1986.

[61] J. F. Sowa, "Conceptual graphs for a data base interface", *IBM Journal of Research and Development*, vol. 20, pp. 336–357, 1976.

[62] K. Pearson, "Contributions to the mathematical theory of evolution. ii. skew variation in homogeneous material", *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 186, pp. 343–414, 1895.

[63] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006.

[64] M. Krzywinski, J. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra, "Circos: an information aesthetic for comparative genomics.", *Genome research*, vol. 19, pp. 1639–45, Sep. 2009.

[65] B. Johnson and B. Shneiderman, "Tree-maps: A space-filling approach to the visualization of hierarchical information structures", in *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, IEEE, 1991, pp. 284–291.

[66] D. Turo and B. Johnson, "Improving the visualization of hierarchies with treemaps: Design issues and experimentation", in *Proceedings of the 3rd Conference on Visualization '92*, ser. VIS '92, Boston, Massachusetts: IEEE Computer Society Press, 1992, pp. 124–131.

[67] Y. Tsuruoka, M. Miwa, K. Hamamoto, J. Tsujii, and S. Ananiadou, "Discovering and visualizing indirect associations between biomedical concepts.", *Bioinformatics (Oxford, England)*, vol. 27, pp. i111–9, Jul. 2011.

[68] D. R. Swanson, "Medical literature as a potential source of new knowledge", *Bull Med Libr Assoc*, vol. 78, no. 1, pp. 29–37, Jan. 1990.

[69] J. Kim, S. So, H. J. Lee, J. C. Park, J. J. Kim, and H. Lee, "DigSee: Disease gene search engine with evidence sentences (version cancer)", *Nucleic Acids Res.*, vol. 41, no. Web Server issue, W510–517, Jul. 2013.

[70] J. Björne, J. Heimonen, F. Ginter, A. Airola, T. Pahikkala, and T. Salakoski, "Extracting complex biological events with rich graph-based feature sets", in *Proceedings of the Workshop*

on Current Trends in Biomedical Natural Language Processing: Shared Task, ser. BioNLP '09, Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 10–18.

[71] R. Bruskiewich, K. Huellas-Bruskiewicz, F. Ahmed, R. Kaliyaperumal, M. Thompson, E. Schultes, K. M. Hettne, A. I. Su, and B. M. Good, "Knowledge.bio: A web application for exploring, building and sharing webs of biomedical relationships mined from pubmed", *BioRxiv*, 2016.

[72] H. Kilicoglu, D. Shin, M. Fiszman, G. Rosemblat, and T. C. Rindflesch, "SemMedDB: a PubMed-scale repository of biomedical semantic predications", *Bioinformatics*, vol. 28, no. 23, pp. 3158–3160, Dec. 2012.

[73] K. M. Hettne, M. Thompson, H. H. van Haagen, E. van der Horst, R. Kaliyaperumal, E. Mina, Z. Tatum, J. F. Laros, E. M. van Mulligen, M. Schuemie, E. Aten, T. S. Li, R. Bruskiewich, B. M. Good, A. I. Su, J. A. Kors, J. den Dunnen, G. J. van Ommen, M. Roos, P. A. 't Hoen, B. Mons, and E. A. Schultes, "The Implicitome: A Resource for Rationalizing Gene-Disease Associations", *PLoS ONE*, vol. 11, no. 2, e0149621, 2016.

[74] C. B. Ahlers, M. Fiszman, D. Demner-Fushman, F. M. Lang, and T. C. Rindflesch, "Extracting semantic predications from Medline citations for pharmacogenomics", *Pac Symp Biocomput*, pp. 209–220, 2007.

[75] K. Sagae and J. Tsujii, "Dependency parsing and domain adaptation with lr models and parser ensembles", in *Proceedings of the CoNLL 2007 Shared Task in the Joint Conferences on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07 shared task)*, Prague, Czech Republic, 2007, pp. 1044–1050.

[76] T. Nunes, D. Campos, S. Matos, and J. L. Oliveira, "Becas: Biomedical concept recognition services and visualization", *Bioinformatics*, vol. 29, no. 15, pp. 1915–1916, 2013. DOI: `10.1093/bioinformatics/btt317`.

[77] K. Degtyarenko, P. de Matos, M. Ennis, J. Hastings, M. Zbinden, A. McNaught, R. Alcantara, M. Darsow, M. Guedj, and M. Ashburner, "ChEBI: a database and ontology for chemical entities of biological interest", *Nucleic Acids Res.*, vol. 36, no. Database issue, pp. D344–350, Jan. 2008.

[78] P. Thompson, J. McNaught, S. Montemagni, N. Calzolari, R. del Gratta, V. Lee, S. Marchi, M. Monachini, P. Pezik, V. Quochi, C. Rupp, Y. Sasaki, G. Venturi, D. Rebholz-Schuhmann, and S. Ananiadou, "The biolexicon: A large-scale terminological resource for biomedical text mining", *BMC Bioinformatics*, vol. 12, no. 1, pp. 1–29, 2011.

[79] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases", in *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, Aug. 2013, pp. 15–19.

[80] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery", *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, Dec. 1983.

[81] RedisLabs. (2016). Introduction to redis, [Online]. Available: `http://redis.io/topics/introduction` (visited on 05/22/2016).

[82] N. Foundation. (2016). About | node.js, [Online]. Available: `https://nodejs.org/en/about/` (visited on 05/23/2016).

[83] I. Fette and A. Melnikov, *The websocket protocol*, RFC 6455 (Proposed Standard), Internet Engineering Task Force, Dec. 2011. [Online]. Available: `http://www.ietf.org/rfc/rfc6455.txt`.

[84] V. G. Cerf and R. E. Kahn, "A protocol for packet network intercommunication.", *Computer Communication Review*, vol. 35, no. 2, pp. 71–82, 2005, Reprint from IEEE Transactions on Communications COM-22, May 1974.

[85] ECMA International, *Standard ECMA-262 - ECMAScript Language Specification*, 6th ed. Jun. 2015.